AD-A051 495 GENERAL ELECTRIC CO ARLINGTON VA F/G 9/2
PREDICTING SOFTWARE COMPREHENSIBILITY.(U)
FEB 78 S B SHEPPARD, M A BORST, L T LOVE N00014-77-C-0158
UNCLASSIFIED TR-78-388100-2 NL

| OF |
AD
A051495

END
DATE
FILMED
4-78
DDC

(12)

TR-77-388100-2

TECHNICAL REPORT

PREDICTING SOFTWARE COMPREHENSIBILITY

FEBRUARY 1978

GENERAL ELECTRIC
INFORMATION SYSTEMS PROGRAMS
ARLINGTON, VIRGINIA

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER TR-388100-2	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) PREDICTING SOFTWARE COMPREHENSIBILITY		5. TYPE OF REPORT & PERIOD COVERED TECHNICAL REPORT 6. PERFORMING ORG. REPORT NUMBER 388100-2
7. AUTHOR(s) S.B. SHEPPARD, M.A. BORST, L.T. LOVE		8. CONTRACT OR GRANT NUMBER(s) N00014-77-C-0158
9. PERFORMING ORGANIZATION NAME AND ADDRESS OFFICE OF NAVAL RESEARCH Arlington, VA 22217		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS NR197-037
11. CONTROLLING OFFICE NAME AND ADDRESS		12. REPORT DATE 2/24/78 13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; Distribution unlimited. Reproduction in whole or in part is permitted for any purpose of the U.S. Government

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

This research was supported by Engineering Psychology Programs, Office of Naval Research

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

MNEMONIC VARIABLE NAMES, STRUCTURED PROGRAMMING, SOFTWARE PSYCHOLOGY METRICS, CONTROL FLOW COMPLEXITY, SOFTWARE ENGINEERING, MODERN PROGRAMMING PRACTICES, PROGRAM MEMORIZATION

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report describes the first experiment in a program of research designed to identify characteristics of computer software which are related to its psychological complexity. This experiment evaluated the effect of three independent variables (mnemonic variable names, complexity of control flow, and general type of program) on a programmer's understanding of a computer program. The contributions of several variables, including Halstead's software science metric and McCabe's complexity metric, to the prediction of program understanding were also evaluated. In a pilot study by Sheppard and Love (1977)

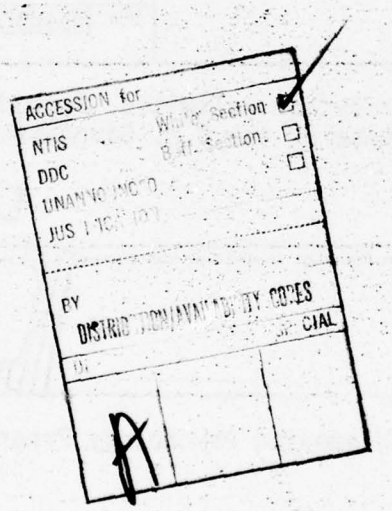
DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601 |

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Block 20 continued

significant results were achieved with the materials and procedures employed here.

Thirty-six experienced programmers were instructed to study a computer program for 20 minutes, and were then given 25 minutes to reconstruct a functionally equivalent one. Performance was measured by the percentage of functionally correct statements recalled. Results indicated that complexity of control flow affected program understanding, while no relationship was found for mnemonic variable names and general program type. The metrics of both Halstead and McCabe were related to program understanding when differences between subjects and specific programs were taken into consideration.



per telecon (14)

TR-78-388100-2

(9) TECHNICAL REPORT

(6) PREDICTING SOFTWARE COMPREHENSIBILITY

by

(10) S.B. Sheppard, M.A. Borst, L.T. Love

(11) February 1978

(12) 68 p.

Submitted to:

Office of Naval Research
Engineering Psychology Programs
Arlington, Virginia 22217

(15) Contract N00014-77-C-0158
Work Unit: NR 197-037

GENERAL ELECTRIC COMPANY
INFORMATION SYSTEMS PROGRAMS
1755 Jefferson Davis Highway
Suite 200
Arlington, Virginia 22202

409 446

PREDICTING SOFTWARE COMPREHENSIBILITY

by

S. B. Sheppard, M. A. Borst,
& L. T. Love

Information Systems Programs
General Electric Company
1755 Jefferson Davis Highway, Suite 200
Arlington, Virginia 22202

February 1978

Software Complexity Research Program

Department of Defense (DOD) software production and maintenance is a large, poorly understood, and inefficient process. Recently Frost and Sullivan (The Military Software Market, 1977) estimated the yearly cost for software within DOD to be as large as $9 billion. DeRoze (1977) has also estimated that 115 major defense systems depend on software for their success. In an effort to find near-term solutions to software related problems, the DOD has begun to support research into the software production process.

A formal 5 year R&D plan (Carlson & DeRoze, 1977) related to the management and control of computer resources was recently written in response to DOD Directive 5000.29. This plan requested research leading to the identification and validation of metrics for software quality. The study described in this paper represents an experimental investigation of such metrics, and is part of a larger research program seeking to provide valuable information about the psychological and human resource aspects of the 5 year plan.

DOD is also initiating the development of a more powerful higher order language for general use by all services (Department of Defense, 1977). With a language independent measure of the complexity of software, we can evaluate not only program A versus program B, but also the individual constructs of a language (cf. Gordon, 1977).

Thus, an objective, quantitative theory based on sound experimental data can replace idiosyncratic, subjective evaluations of the psychololgical complexity of software. Long term benefits of this effort involve improved software system reliabilty and reduced development and maintenance costs.

The challenge undertaken in this research program is to quantify the psychological complexity of software. It is important to distinguish clearly between the psychological and computational complexity of software. Computational complexity refers to characteristics of algorithms or programs which make their proof of correctness difficult, lengthy, or impossible. For example, as the number of distinct paths through a program increases, the computational complexity also increases. Psychological complexity refers to those characteristics of software which make human understanding of software more difficult. No direct linear relationship between computational and psychological complexity is expected. A program with many control paths may not be psychologically complex. Any regularity to the branching process within a program may be used by a programmer to simplify understanding of the program.

Halstead (1977) has recently developed a theory concerned with the psychological aspects of computer programming. His theory provides objective estimates of the

effort and time required to generate a program, the effort required to understand a program, and the number of bugs in a particular program (Fitzsimmons & Love, in press). Some predictions of the theory are counterintuitive and contradict some results of previous psychological research. The theory has attracted attention because independent tests of hypotheses derived from it have proven amazingly accurate.

Although predictions of programmer behavior have been particularly impressive, much of the research testing Halstead's theory has been performed without sufficient experimental or statistical controls. Further, much of the data were based upon imprecise estimating techniques. Nevertheless, the available evidence has been sufficient to justify a rigorous evaluation of the theory.

Rather than initiate a research program designed specifically to test the theory of software science, a research strategy was chosen which would generate suggestions for improving programmer efficiency regardless of the success of any particular theory. This research focuses on four phases of the software life-cycle: understanding, modification, debugging, and construction. Since different cognitive processes are assumed to predominate in each phase, no single experiment or set of experiments on a particular phase would provide sufficient basis for making broad recommendations for improving

programmer efficiency. Each experiment in the series comprising this research program has been designed to test important variables assumed to affect a particular phase of software development. Professional programmers will be used in these experiments to provide the greatest possible external validity for the results (Campbell & Stanley, 1966). In addition, Halstead's theory of software science and other related metrics can be evaluated with these data.

ACKNOWLEDGEMENTS

The authors gratefully acknowledge the assistance of Dr. Bill Curtis in analyzing the data and in revising this technical report. We anticipate a long and productive association with him as a member of our staff. We also appreciate the assistance of Dr. Gerald Hahn of General Electric's Corporate Statistics Staff in developing the experimental design. Dr. John O'Hare's careful review of a preliminary version of this report has resulted in substantial improvements.

TABLE OF CONTENTS

Predicting Software Comprehensibility

ABSTRACT

This report describes the first experiment in a program of research designed to identify characteristics of computer software which are related to its psychololgical complexity. This experiment evaluated the effect of three independent variables (mnemonic variable names, level of program structure, and general type of program) on a programmer's understanding of a computer program. The contributions of several variables to the prediction of program understanding were also evaluated. Significant results were achieved in a pilot study by Sheppard and Love (1977) using the materials and procedures employed here.

Thirty-six experienced programmers were instructed to study a computer program for 20 minutes, and were then given 25 minutes to reconstruct a functionally equivalent program. Performance was measured by the percentage of functionally correct statements recalled. Results indicated that level of program structure and program class affected program understanding, while no relationship was found for mnemonic variable names. The metrics of both Halstead and McCabe were related to program understanding when differences between subjects and specific programs were taken into consideration.

Predicting Software Comprehensibility

1.0 INTRODUCTION

Programmers' ability to understand computer programs may have substantial impact on their efficiency in debugging or modifying these programs. There are several software engineering practices which have been designed to increase programmers' efficiency in terms of both the accuracy and speed of their work. Programs developed in accordance with these practices should be more easily understood.

Dijkstra (1972) suggested that program construction should proceed in a top-down structured fashion. He contended that structured programs are easier to understand, debug, and modify. In a study using student programmers and text book programs, Love (1977) found that simplified control flow made programs easier to understand for graduate (but not for introductory) students. That study did not use programs which were strictly structured.

Another standard software engineering practice is the use of carefully chosen variable names which serve as mnemonic aids in understanding programs. Weissman's (1974) research suggested that menmonic variable names resulted in performance increases (up to a factor of 2). His results need replication since there were difficulties with his experimental design and dependent measures.

In parallel with these attempts to improve programmer efficiency, several approaches were developed for predicting the psychological complexity of software algorithms. In

1972, Halstead first published his software physics theory (later renamed software science) stating that algorithms have measurable characteristics analogous to physical laws. His objective was to develop quantitative measures of the complexity of computer programs in terms of language level, algorithm purity, programming effort, and programming time. Preliminary tests of the theory have shown very high correlations (some greater than .90) between his software physics metrics and such dependent measures as the number of bugs in programs (Funami & Halstead, 1975), programming time (Gordon & Halstead, 1975), and quality of programs (Halstead, 1973).

There have been several recent attempts to develop metrics for the complexity of control flow through a computer program (e.g., Bell & Sullivan, 1974). One of the most promising of these metrics was proposed by McCabe (1976). McCabe's metric will be used in this study as an alternative against which Halstead's metrics can be compared.

A critical issue in assessing the utility of these software engineering practices and metrics involves the definition of a dependent variable. A model of a programmer's understanding of a computer program is shown in Figure 1. First, a programmer must understand the overall purpose of a program. Then an interactive process begins in which successive modules must be understood separately, and

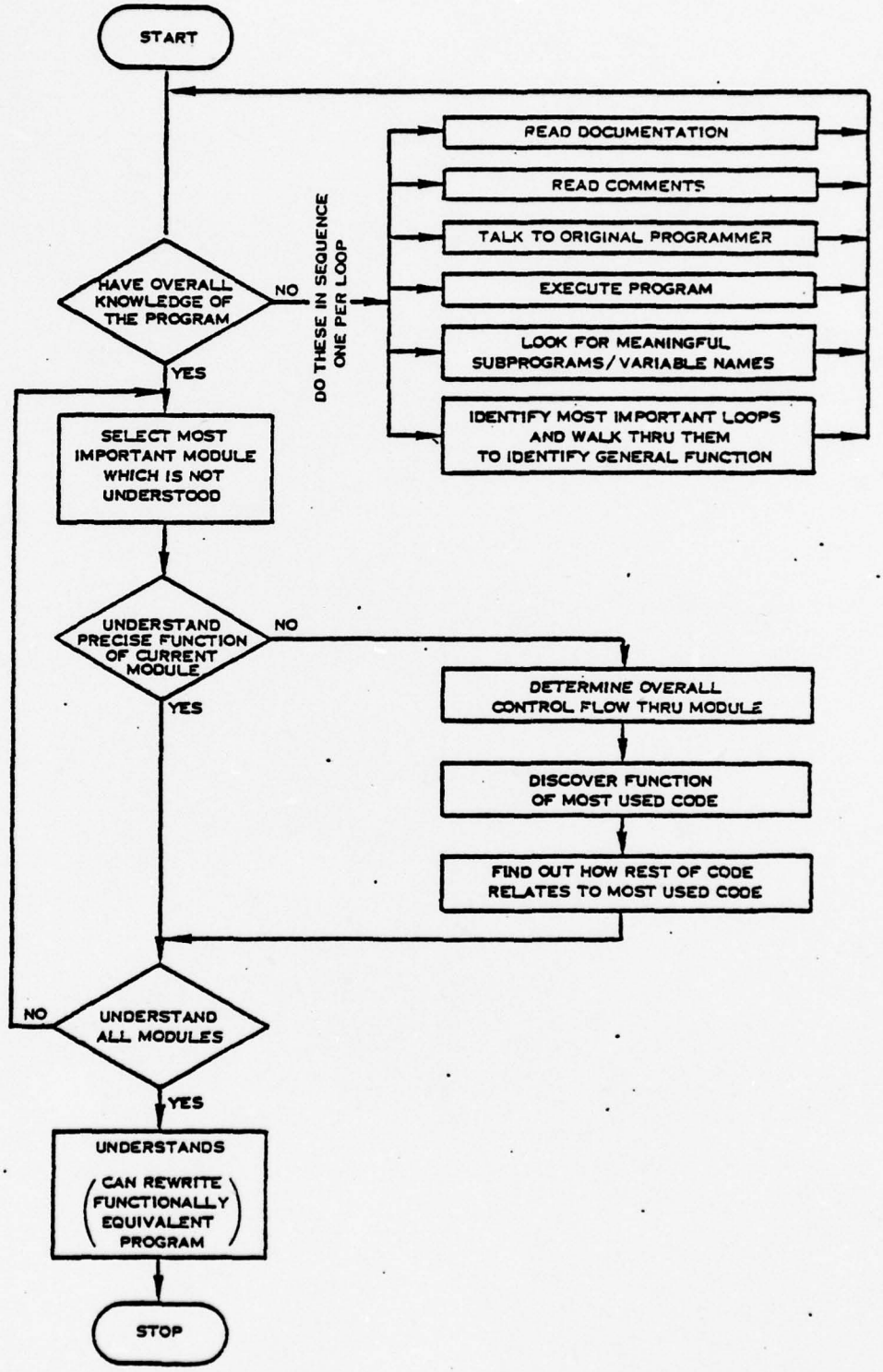


Figure 1: MODEL OF HUMAN UNDERSTANDING OF SOFTWARE

then integrated into the overall flow of the program. Measures of understanding such as quiz scores or ability to hand simulate a program may have reflected existing knowledge of programming methods and techniques rather than specific knowledge of the particular program under consideration.

Current literature (Love. 1977; Shneiderman, 1974, 1977) suggests that the most sensitive measure of whether people understand a computer program is their ability to learn a program's structure and reproduce a functionally equivalent program without notes. It would be extremely difficult to reproduce a non-trivial program without some understanding of its function. The dependent measure employed here was the functional correctness of a participant's reconstructed program.

The main purpose of this experiment was to ascertain the relationship between two programming style variables and the ability to understand a program. There was also an assessment of whether comprehensibility differed as a function of program type. In addition, the relationship between comprehensibility and three program metrics (i.e., Halstead's E, McCabe's V(G), and the number of statements) was evaluated.

2.0 METHOD

2.1 Participants

Thirty-six professional programmers were tested in five different locations. Each participant was a General Electric employee with working knowledge of FORTRAN. These programmers had an average of 6.8 years of professional programming experience (ranging from 0 to 18 years). The majority (n=23) came from an engineering background, two were statistical programmers, and nine had been primarily involved with non-numeric or text processing software.

2.2 Procedure

A packet of materials was prepared for each participant. The initial instructions to each participant are presented in Appendix 6.1. The written instructions included questions on the extent of programming experience and area of expertise. The first exercise was a short FORTRAN program with a brief description of its purpose All 36 participants received the same program, which they were allowed to study for ten minutes. They were permitted to make notes or draw flowcharts. At the end of the study period, the original program and all scrap papers were collected. Each participant was then given five minutes to reconstruct a functional equivalent of the program from memory on a blank sheet of paper, but was not required to reproduce the comment section.

The purposes of this short introductory program were

1) to provide a common basis for comparing the skills of the participants on this type of task, and

2) to control for initial learning effects.

This second point is important since a previous study (Sheppard and Love, 1977) indicated that learning may occur during an initial task of this type.

Following this initial exercise, participants were presented in turn with three programs which comprised the experimental task for this study. They were allowed 25 minutes for study and 20 minutes for the reconstruction of each program. A break of 15 minutes occurred before the last program was presented.

2.3 Experimental Design

In order to control for individual differences in performance, a within-subjects 3^4 fractional factorial design was employed in this experiment (Hahn & Shaprio, 1966). Nine programs of three general classes were tested (Table 1). Three levels of program structure were defined for each of the nine programs, and each of these 27 versions was presented in three levels of variable mnemonicity for a total of 81 programs. The programs were selected from a set of programs solicited from practicing programmers at several GE locations.

Four sets of nine participants were used in the experiment. The 27 participants in the first three sets exhausted the total of 81 programs. The fourth set of 9

TABLE 1
EXPERIMENTAL DESIGN

PROGRAM		UNSTRUCTURED			QUASI-STRUCTURED			STRUCTURED			— CONTROL FLOW
CLASS	NAME	LO	MED.	HIGH	LO	MED.	HIGH	LO	MED.	HIGH	— MNEMONICS
STATISTICS	CHISQ	1	5	9	4	8	3	7	2	6	
	PHI	10	14	18	13	17	12	16	11	15	
	TOTAL	19	26	24	22	20	27	25	23	21	
ENGINEERING	BESSEL	3	4	8	6	7	2	9	1	5	
	FOURIER	11	15	16	14	18	10	17	12	13	
	INTEG	20	27	22	23	21	25	26	24	19	
NON-NUMERIC	SELECT	2	6	7	5	9	1	8	3	4	
	UNIQUE	12	13	17	15	16	11	18	10	14	
	WIDTH	21	25	23	24	19	26	27	22	20	

NOTE: EACH NUMBER REPRESENTS THE ASSIGNMENT OF ONE PARTICIPANT WITHIN THE BLOCK OF 27 PARTICIPANTS (3 SETS OF 9 PARTICIPANTS EACH).

participants repeated one of the three previous sets. Table 1 shows the design for the first 27 participants.

Programmers at each location were randomly assigned to experimental conditions in order that over the course of their three experimental programs, every participant had worked with a program from each class, and at each level of structure and variable mnemonicity. For example, participant 4 received the following three programs: 1) BESSEL - an engineering program, unstructured control flow, medium mnemonic level, 2) CHISQ - a statistical program, quasi-structured control flow, least mnemonic level, and 3) SELECT - a non-numeric program, structured control flow, most mnemonic level. For simplicity the design is presented in Table 1 without regard for the order of presentation to the participants. One of the six possible orders of presentation of three programs was assigned randomly and without replacement to each participant.

2.4 Independent Variables

2.4.1 Program Class

Three general classes of programs were used: engineering, statistical, and non-numeric. Three programs of each class were developed. Appendix 6.2 describes the purpose of the nine programs and shows their lengths, which varied from 36 to 57 statements. The programs selected were considered to be representative of the type of programs actually encountered by professional programmers in each of

these areas. Appendix 6.3 has some sample program listings. All nine programs used in this experiment were compiled and executed using appropriate test data.

4.2 Program Structure

Three levels of program structure were defined. The structured level adhered strictly to the tenets of structured programming (Dijkstra, 1972). Program flow proceeded from top to bottom with one entry and one exit. Neither backward transfer of control nor arithmetic IF's were allowed.

Awkward constructions may occur in FORTRAN, when the rules for structured programming are applied rigorously. These include necessary but artificial GO TO's and DO loops with dummy variables (Tenny, 1974). These awkward constructions were largely eliminated in the quasi-structured level, where a more natural control flow was allowed. A judicious use of backward GO TO statements and multiple exits was permitted. IF statements were again restricted to assignment and logical IF's.

In the unstructured version of each program the control flow was not straightforward. The GO TO statement occurred more frequently and backward transfer of control was not restricted. The three-way transfer of control statement (arithmetic IF) was allowed only at this level (Table 2).

Two well-known metrics for program complexity were also calculated for each of the programs in order to determine

TABLE 2

CONTROL STRUCTURES ALLOWED
IN THE
THREE LEVELS OF COMPLEXITY

STRUCTURED	QUASI-STRUCTURED	UNSTRUCTURED
DO LOOP FORWARD EXIT ONLY	DO LOOP BACKWARD EXIT ALLOWED	→
		EXPANDED DO LOOP
LOGICAL IF →	→	→
		ARITHMETIC IF IF ( ) -, 0, +
GO TO FORWARD ONLY	GO TO BACKWARD ALLOWED →	→
COMPUTED GO TO FORWARD ONLY - SINGLE ENTRY AND EXIT	COMPUTED GO TO → BACKWARD ALLOWED	→
		ASSIGNED GO TO
SINGLE RETURN	MULTIPLE RETURNS ALLOWED →	→

their correspondence with the working definitions. The metrics selected were: McCabe's V(G) (McCabe, 1976) and Halstead's E (Halstead, 1977).

2.4.2.1 Halstead's E

In Halstead's theory of software science, the amount of effort required to generate a program (E), can be calculated from simple counts of the actual code. The calculations are based on four quantities: 1) the number of distinct operators and operands, and 2) the number of occurrences of operators and operands. From these relationships, Halstead derives the number of mental comparisons required to generate a program.

Since different programming languages produce varying numbers of instructions, the number of elementary mental discriminations for each mental comparison varies with the language used. When a correction is made to account for these differences, one can define E in terms of the number of mental discriminations per program; i.e., the number of comparisons in the program multiplied by the average number of mental discriminations made per comparison. A discussion of the computational formula can be found in Fitzsimmons and Love (in press) or Halstead (1977).

All software science metrics were computed precisely from a program (based on Ottenstein, 1975) which had as input the source code listings of 27 programs (9 separate programs at each of three levels of structure).

2.4.2.2 McCabe's V(G)

McCabe's metric is the classical graph-theory cyclometric number, defined as V(G) = # edges - # nodes + # connected regions. Because the McCabe measure is defined only for programs that adhere strictly to the rules of structured programming, some modifications to the metric were necessary in order to evaluate the less structured control flow versions. (See Appendix 6.4 for a description of these modifications). All experimental programs were checked before the experiment to insure that the most complex version of program had the highest McCabe value and the least complex version had the lowest value.

2.4.3 Variable Name Mnemonicity

Three levels of mnemonicity for variable names were manipulated independently of program structure levels. Because meaningfulness is difficult to assign arbitrarily, a preliminary assessment was done. The nine programs were modified so that the variable names were V1, V2, ..., VN. Professional programmers were presented the programs and descriptions of their purpose. They were asked to substitute meaningful names for the 'V' names. The names most often generated were used in the most mnemonic condition. The moderately mnemonic level consisted of less frequently chosen names. In the least mnemonic condition names consisted of a randomly chosen letter such that all real variables began with A through H or O through Z, and

all integer variables began with I through N. In the few cases where there were more than six integer names, the letter was followed by a single digit (e.g., I2). Counters in DO loops often had idential names in all three mnemonic versions (See Appendix 6.5), since long mnemonic variable names are rarely used as counters in programs.

2.5 Covariates

In order to obtain a measure which was assumed to be related to programming ability, all participants were required to perform the same preliminary task. A short program was given to each participant to study and then reconstruct. Their scores on this task were used as a covariate to measure individual performance differences. Participants were also asked their type of programming experience and the number of years they had been programming professionally. Situational covariates included the sequence of presentation and the specific program.

2.6 Dependent Variable

All warm-up programs were scored by the same grader. The remaining 108 experimental programs were scored independently by three graders. The criterion for scoring the programs was the functional correctness of each separately reconstructed statement. Variable names and statement numbers which differed from those in the original program were counted as correct when used consistently. All errors were classified as either syntactical or logical.

Only one error of each type was counted per statement, even though multiple syntactical and logical errors could occur in the same statement. Control structures could be different from the original program as long as the statement performed the same function.

Because it is difficult to prove the equivalence of two versions of the same program, function, or statement, three judges scored each program independently. Interjudge correlations of .96, .96, and .94 were obtained across the three sets of scores. The average of the three scores (percents of statements correctly reconstructed) for each program was used as the dependent variable in the data analysis.

2.7 Analysis

The analysis of results was conducted in two phases. The first phase was an experimental test of the programming style variables, while the second phase was an evaluation of the software complexity metrics.

The first phase, involving an experimental test of programming practices, was analyzed in a hierarchical regression analysis. In this analysis, domains of variables were entered sequentially into a multiple regression equation to determine if each successive domain significantly improved the prediction of the equation developed from domains already entered. Thus, the order with which domains were entered into the analysis was

important. In this study effects related to pre-existing differences among participants and programs were entered into the analysis prior to evaluating the effects of programming styles. The variable domains were entered in the following order:

Differences related to participants and programs

1) Pretest scores
2) Class of program
3) Specific program

Programming styles

4) Program structure
5) Variable mnemonicity
6) The interaction between program structure and variable mnemonicity.

The variables representing the different conditions of domains 2 through 5 were effect coded (Kerlinger & Pedhazur, 1973).

The second phase of analysis investigated relationships among Halstead's E, McCabe's V(G), number of statements in the program, and performance. Analysis consisted of examining correlations among the measures in both the raw data and data corrected for differences among participants and programs.

3.0 RESULTS

3.1 Individual Differences among Participants

Data presented in Table 3 indicate that scores on the pretest were significantly related to the percent of statements correctly reconstructed during the experiment. Pretest scores accounted for 17% of the variance in performance, while no relationships were observed for type or length of programming experience. The two statistical programmers recalled more statements correctly than engineering or non-numeric programmers, but generalization is not possible from such a limited sample. Further, job location was not related to performance.

3.2 Differences among Programs

A mean of 50% of the statements were correctly recalled across all programs and experimental conditions. While this was a preferred level for mean task difficulty, there were substantial differences in difficulty among the various programs. As evident in Table 3, performance differed significantly as a function of the class of program. These differences accounted for 8% of the variance in performance in addition to that accounted for by individual differences among participants. Engineering programs were the most difficult (41% of the statements correctly recalled), followed by statistical (52%) and non-numeric (57%).

When the specific program was taken into account an additional 20% of the variance in performance was

TABLE 3

HIERARCHIAL REGRESSION ANALYSIS

VARIABLE DOMAIN	DOMAIN[a] R^2	df	ΔR^{2b}
1) PRETEST	.17**	1	.17**
2) CLASS OF PROGRAM	.09**	2	.08**
3) SPECIFIC PROGRAM	.26**	8	.20**
4) PROGRAM STRUCTURE (PS)	.07**	2	.07**
5) VARIABLE MNEMONICITY (VM)	.01	2	.01
6) PS X VM	.03	4	.03
TOTAL		19	.56

Note: n = 108

a Correlations in this column represent the total relationship between each variable domain and performance. Where there is only one degree of freedom for a particular domain, figures in this column represent zero-order correlations, otherwise they represent multiple correlations for all variables in the domain.

b Figures in this column indicate the percent of variance contributed to prediction of performance in addition to that afforded by preceding domains. Significance levels indicate whether this represented a significant contribution to prediction.

** $p \leq .01$

explained.However, this result is not strictly a function of differences among programs, because variance related to specific programs was confounded with variance related to participants. Overall, 45% of the variance in performance was accounted for by differences among participants and general program characteristics.

3.3 Program Structure

Significant differences in performance were obtained as a function of program structure. The three levels of structure accounted for 7% of the variance in performance in addition to variance related to differences among programs and participants. As expected, the least structured level was the most difficult to reconstruct (Figure 2). Contrary to the tenets of structured programming, however, the most structured level did not produce the best performance. A greater percent of statements were recalled from quasi-structured programs, conceivably because of their less cumbersome constructs. A post hoc analysis (Scheffe, 1959) showed the means for the quasi- and unstructured programs to be significantly different ($p \leq .05$).

3.4 Variable Name Mnemonicity

No significant differences in performance were observed in relation to the three levels of mnemnonicity assigned to variable names. Consequently, variable mnemonicity did not contribute significantly to the hierarchical regression equation. Further, no significant interaction was found

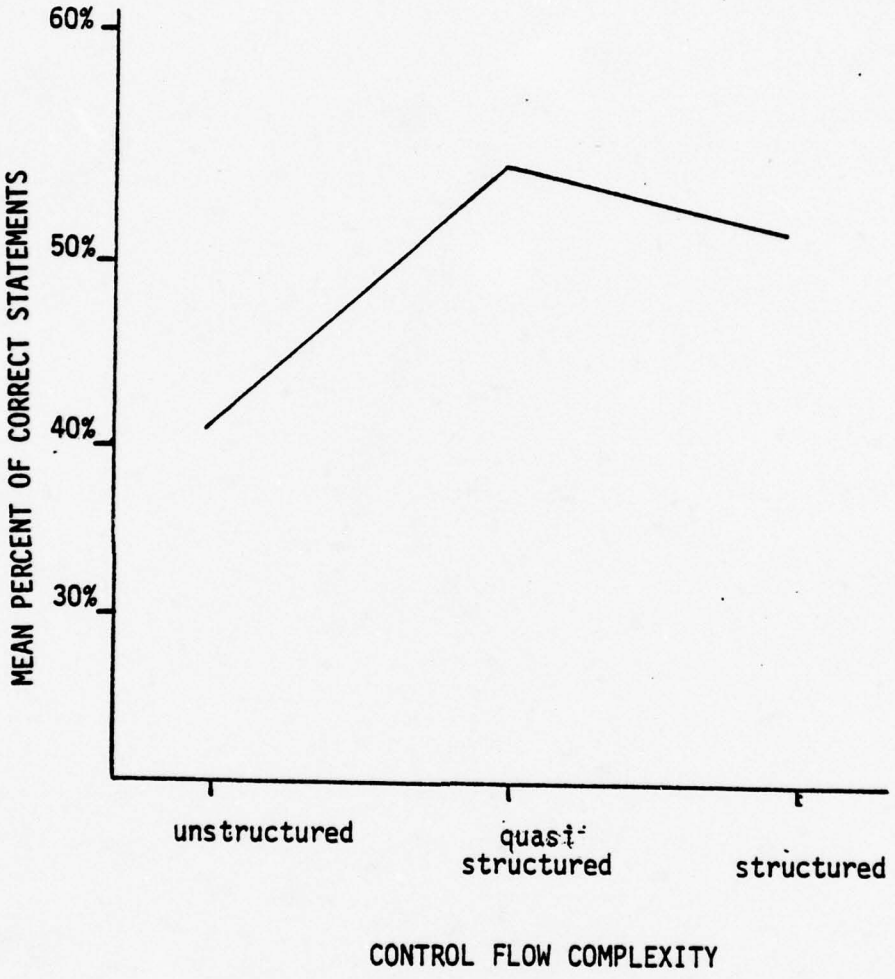


FIGURE 2: Mean percent of statements correctly recalled for three levels of program structure

between variable mnemonicity and level of structure.

3.5 Order of Presentation

Performance did not differ as a function of the order in which the programs were presented to participants, suggesting that any learning process which might have affected the results occurred during the pretest rather than during the three experimental tasks.

Since different levels of variable mnemonicity neither affected performance significantly, nor caused any change in complexity metrics for a particular program, the data reported in this section were averaged over levels of mnemonicity. Thus, the 27 data points each represent a value for a specfic program at a specific level of structure. This averaging process also reduced to some extent the effect of individual differences among participants since each data point is averaged across either 3 or 6 participants (9 of the conditions were repeated by an additional three participants).

3.6.1 Relationships among Metrics

Table 4 presents correlations among the three metrics of software complexity employed in this study; namely, length (number of statements), McCabe's V(G), and Halstead's E. Length and McCabe's V(G) were strongly correlated, while Halstead's E displayed only moderate relationships with these other two metrics. Investigation of the scatterplots (Appendix 6.6) indicated that the correlations for

TABLE 4

Correlations among Metrics of Software Complexity

METRIC	CORRELATIONS	
	LENGTH	MCCABE
MCCABE	.75**	
HALSTEAD	.47**	.42*
HALSTEAD (n=26)	.75**	.84**

NOTE: Except where indicated, n = 27.

*$p \leq .05$
**$p \leq .01$

Halstead's E were weakened by an extreme value for E of 250.With this outlier removed the correlations of E with the other two metrics rose to the same level observed in their intercorrelation.

3.6.2 Relationships of Metrics with Performance

The percents of variance in performance accounted for by each complexity metric are presented in Table 5. The correlations underlying the data reported in this section were all negative indicating that performance fell as the level of complexity indexed by these three metrics increased. Length and McCabe's V(G) were moderately related to performance, accounting for 28% and 20% of the variance, respectively. Halstead's E was not significantly related to performance. Investigation of the scatterplot for the Halstead result (Figure 3), however, produced some interesting observations. There were three data points (circled in Figure 3) which were developed by averaging across three participants who consistently outscored other participants on both the pretest and the experimental programs. Recognizing the effect of individual differences on performance, it was likely that the three data points generated by this group of participants resulted more from the failure of random assignment to fully neutralize the effect of individual differences, than from the experimental conditions. When these three data points were removed the variance accounted for by all three metrics increased.

TABLE 5

Percent of Variance in Performance Accounted for by Software Complexity Metric

METRIC	PERCENT OF VARIANCE: ALL GROUPS (n=27)	PERCENT OF VARIANCE: EXCEPTIONAL GROUP REMOVED (n=24)
RAW SCORES		
LENGTH	.28**	.37***
MCCABE'S V(G)	.20**	.26**
HALSTEAD'S E	.02	.13*
TRANSFORMED SCORES		
LENGTH	.14*	.42***
MCCABE'S V(G)	.31***	.48***
HALSTEAD'S E	.04	.53***

*$p \leq .05$
**$p \leq .01$
***$p \leq .001$

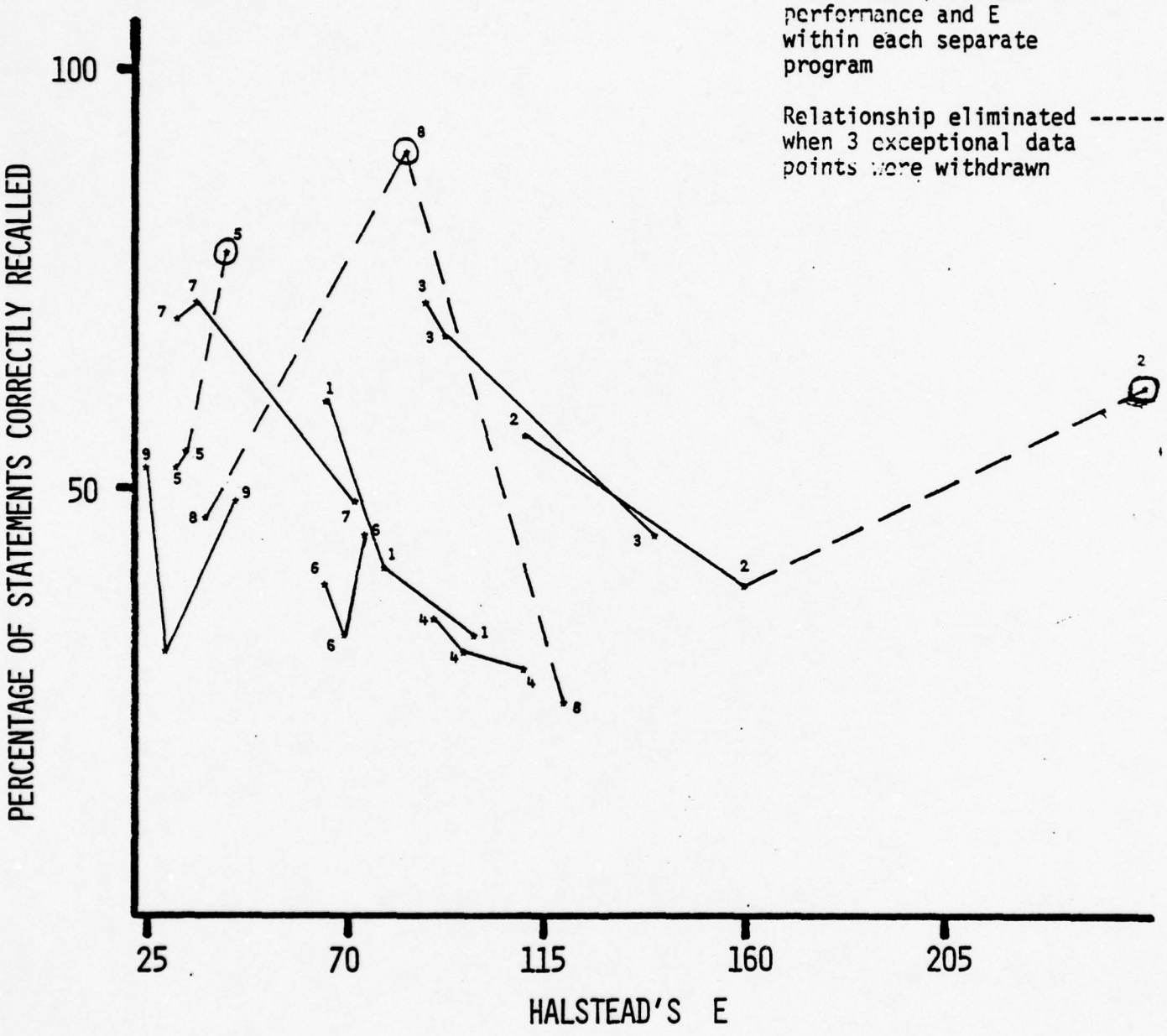


FIGURE 3: PERCENT OF STATEMENTS CORRECTLY RECALLED VERSUS HALSTEAD'S E

NOTE: NUMBERS IN THE FIGURE REPRESENT THE SPECIFIC PROGRAM

With the three points for the exceptional group removed, the relationships between performance and the complexity metrics were generally linear within the data for each specific program (observe the solid lines in Figure 3). It was apparent from Figure 3, and was confirmed in the regression analyses, that there were considerable differences among programs in difficulty and complexity. In order to determine whether the complexity metrics were more predictive of performance within programs than across them, a transformation was applied to the data. The lowest value obtained for each metric among the three versions of each specific program was set to zero. Similarly, the percent of statements correctly recalled for the version with the lowest value of the particular metric was also set to zero, allowing the percent recalled to fall below zero in several instances (a difference score). This transformation of scores represented an attempt to determine whether performance diminished as a function of increasing complexity when initial differences among programs had been removed.

The percents of variance in performance accounted for by each of the transformed metrics are presented in Table 5 for samples with and without the exceptional group of participants removed. The effect of the transformation on the results for length was not great. However, substantial increases were observed for both McCabe's V(G) and

Halstead's E. Thus, while Halstead's E accounted for only 2% of the variance in performance among the raw scores, it accounted for 53% after some corrections were made for differences among programs and participants.

4.0 DISCUSSION

Three factors were found to influence programmers' ability to correctly recall programs they had previously studied. These factors included individual differences among participants, characteristics of specific programs, and the level of program structure. Each of these factors contributed separately to the prediction of performance on the task studied here. In addition, several metrics of software complexity were found to predict understanding under certain conditions.

Individual differences among programmers represent an important topic which has been mentioned more frequently than studied. Such differences (as measured by a pretest) accounted for almost one-fifth of the variance in performance. The effect of individual differences might be even greater were the sample expanded beyond the experienced, professional programmers studied here. The size of the effect for these differences suggests that an important area for future research will be in identifying strategies for the selection, placement and training of computer programmers.

Performance effects due to differences among programs are not easily explained from these results. While there was a significant effect due to class of program (engineering vs. non-numeric vs. statistical), this result may have been a function of some experiencial or familarity

factor particular to the sample studied. Further, effects due to differences among specific programs were confounded in this experimental design with effects related to individual differences among participants. Nevertheless, the suggestion of substantial differences in the ease of understanding based on type of program is interesting in light of the limited range of programs employed here.

In order to properly analyze subtle participant X program interactions, more participants would be required in each cell of this experimental design. Such effects may involve interactions between the nature or purpose of the program and some experiential or cognitive factor among programmers. Such interactions may hold important implications for the management of software development projects, especially regarding the assignment of programmers.

The characteristics of programs studied in this experiment included two principles of programming style and several metrics of software complexity. Of the programming style variables examined, only level of structure had an effect on performance. Results confirmed that well structured programs were easier to understand than unstructured ones. Yet, adhering strictly to the rules of structured programming in FORTRAN often caused clumsy constructions that were no more effective than taking certain liberties with the structure to create cleaner

code.These liberties included multiple returns and judiciously placed backward GO TO statements; both are violations of the rules for structured coding.

No differences in performance were observed as a function of the mnemonic value of variable names. However, many participants evidenced a preference for mnemonic names, in that they used their own, more meaningful names when rewriting the least mnemonic versions of the programs. For the medium and most mnemonic versions they tended to use the names supplied in the original programs. Thus, the importance of mnemonic variable names is supported by the anecdotal rather than statistical evidence.

In addition to the experimental evidence that the level of program structure affects understanding, there was correlational evidence that understanding was related to the psychological complexity of the program. The best predictor of performance in the raw data among the three complexity metrics studied was the total number of statements in the program. However, the considerable amount of variance in this study related to differences among individuals and programs may have masked relationships between performance and complexity metrics, especially Halstead's E. When the data were transformed in an attempt to remove some of the effects due to differences among programs and exceptional programmers, the Halstead and McCabe metrics were found to be substantially better predictors of performance than they

had been in the untransformed data. This result is consistent with the finding in a pilot study (Sheppard & Love, 1977) that Halstead's E was highly correlated with performance.

In essence, the relationships of the Halstead and McCabe metrics with performance were approximately linear within different versions of a single program. This observation implies that prediction of the number of bugs in a program by the Halstead and McCabe metrics alone may not prove adequate. Prediction may be substantially improved by including some variable(s) relating to differences among programs or programmer X program interactions. This issue will be addressed by future studies in this research program.

An interesting problem in the Halstead metric was observed in a program which generated a Halstead value of 250 and a McCabe value of only 6. Inspection of the program showed a series of assignment statements of the form:

```
IF (AMT2.LT.AMT3) MAXPHI = SQRT((AMT2/AMT1) + (AMT4/AMT3))
```

Although such statements resulted in a high E value, they did not affect the control flow. Relationships between performance and different ways of computing complexity metrics will be addressed as data are collected in additional experiments from this research program. These additional data will provide a more comprehensive base from which to test hypotheses regarding these metrics.

Finally, it should be noted that smaller, single factor experiments would have precluded analysis of a number of the effects reported in this study. In particular, implications that individual and/or program factors may need to be included in predictions of psychological complexity may not have emerged had this study included only three rather than nine separate computer programs (cf. Sheppard & Love, 1977).

5.0 REFERENCES

Amster, S.J., Davis, E.J., Dickman, B.N., & Kouni, J.P. An experiment in automatic quality evaluation of software. Proceedings of the Symposium on Computer Software Engineering, 1976, 24, 171-197.

Bell, D.E., & Sullivan, J.E. Further investigations into the complexity of software (Tech. Rep. MTR-2874). Bedford,Mass.: MITRE Corp., June 1974.

Campbell, D., & Stanley, J.C. Experimental and quasi-experimental designs for research. Chicago: Rand McNally, 1976.

Carlson, W.E., & DeRoze, B. Defense system software research and development plan. Unpublished manuscript, Arlington, VA.: Defense Advanced Research Projects Agency, September 1977.

Department of Defense requirments for high order computer programming languages: Revised "IRONMAN". Sigplan Notices, 1977, 12, 39-54.

DeRoze, B. Software research and development technology in the Department of Defense. Paper presented at the AIIE Confererence on Software, Washington, D.C.: December 1977.

Dijkstra, E.W. Notes on structured programming. In O.J. Dahl, E.W. Dijkstra, & C.A.R. Hoare (Eds.), Structured programming. New York: Academic Press, 1972.

Fitzsimmons, A.B.,& Love, L.T. A review and evaluation of software science. ACM Computing Surveys, in press.

Funami, Y., & Halstead, M.H. A software physics analysis of Akiyama's debugging data (Tech. Rep. CSD-TR-144). West Lafayette, Ind.: Purdue University, Computer Science Department, May 1975.

Gordon, R.D. A measure of mental effort related to program clarity. Unpublished doctoral dissertation, Purdue University, 1977.

Gordon, R.D.,& Halstead, M.H. An experiment comparing FORTRAN programming time with the software physics hypothesis. AFIPS Conference Proceedings, 1976, 45, 935-937.

Hahn, G.J.,& Shapiro, S.S. A catalogue and computer program for the design and analysis of orthogonal symmetric and asymmetric fractional factorial experiments (Tech. Rep. 66-C-165). Schenectady, N.Y.: General Electric, May 1966.

Halstead, M.H. An experimental determination of the "purity" of a trivial algorithm (Tech. Rep. 73). West Lafayette, Ind.: Purdue University, Computer Science Department, 1973.

Halstead, M.H. Software physics: Basic principles (Tech. Rep. RJ-1582). Yorktown Heights, NY.: IBM, 1975.

Halstead, M.H. Elements of software science. New York: Elsevier North-Holland, 1977.

Kerlinger, F.N., & Pedhauzer, E.J. Multiple regression in behavioral research. New York: Holt, Rinehart & Winston, 1973.

Love, L.T. Relating individual differences in computer programming performance to human information processing abilities. Unpublished doctoral dissertation, University of Washington, 1977.

McCabe, T.J. A complexity measure. IEEE Transactions on Software Engineering, 1976, SE-2, 308-320.

Ottenstein, K.J. A program to count operators and operands for ANSI-FORTRAN modules (Tech. Rep. CSD-TR-196). West Lafayette, Ind.: Purdue University Computer Science Department, June 1976.

Ramsey, H.R. Human factors in computer systems: A review of the literature (Tech. Rep. SAI-77-054-DEN). Englewood, Col.: Science Applications, April 1977.

Reiter, R.W. Measuring software complexity: An experimental study. Unpublished manuscript, University of Maryland, Department of Computer Science, 1977.

Richards, P. Localization of variables: A measure of program complexity (GE-TIS-76CIS07). Sunnyvale, Calif.: General Electric, December 1976.

Scheffe, H.A. The analysis of variance. New York: Wiley, 1959.

Sheppard, S.B., & Love, L.T. A preliminary experiment to test influences on human understanding of software. Paper presented at the meeting of the Human Factors Society, San Francisco, October 1977.

Shneiderman, B. Human factors experiments in programming: Motivation, methodology, and research directions (Tech. Rep. ISM-TR-9). College Park, Md.: University of Maryland, Department of Information Systems Management, September 1976.

Shneiderman, B. Measuring computer program quality and comprehension (Tech. Rep. ISM-TR-16). College Park, MD.: University of Maryland, Department of Information Systems Management, February 1977.

Sullivan, J.E. Measuring the complexity of computer software (Tech. Tep. MTR-2648). Bedford, MA: MITRE Corp., June 1973.

Tenny, T. Structured programming in FORTRAN. Datamation, 1974, 20, 110-115.

The Military software market (Rep. 427). New York: Frost & Sullivan, 1977.

Weissman, L.M. A methodology for studying the psychological complexity of computer programs (Tech. Rep. TR-CSRG-37). Toronto, Canada: University of Toronto, Computer Systems Research Group, 1974.

INSTRUCTIONS TO PARTICIPANTS

GOOD MORNING!!

Today we are going to ask you to participate in an experiment which we hope will be both entertaining and challenging.

This work, sponsored by the Office of Naval Research, is being done to make computer programs more readable. To do this, we need to measure your understanding of a particular program. We will give you three separate programs and ask you to study each program carefully; then reconstruct that program from memory without any notes.

In previous research, we have discovered that recoding a program from memory is a very sensitive measure of understanding of a program.

Our purpose is to evaluate different ways of writing a computer program. It is not to evaluate computer programmers. Your performance on a program will be compared only to your performance on other programs. Your only competition is yourself. All programs and papers that you will be handed are carefully numbered so it is not necessary for you put your name on any of these.

We would like you to answer the following questions for our research purposes:

1. How long have you been programming in FORTRAN professionally?
_____years _____months

2. Please circle one of the following: Has your primary experience been in Engineering, Statistical or Non-numeric programs?

During this experiment, each of you will be working on different programs. If someone else seems to finish earlier than you, don't be concerned. They will have been working on something else entirely which might not require as much time.

We will begin this morning with a simple test program. We will ask you to study this FORTRAN program for ten minutes. During this time, you may write anything, draw a flow chart, or make any notes to help you understand and memorize the program. When the 10 minutes are up, you will be asked to hand in the programs and any notes. We will then give you 5 minutes to rewrite the program from memory as best you can. Since we are interested in your understanding of the program, it it not necessary for you to memorize statements, statement numbers, or variable names exactly. It's O.K. as long as the program still does the same job.

If there are any questions, please ask them at this time.

Appendix 6.2.1 - A Description of the Statistical Programs

PROGRAM #	CHISQ 1			PHI 2			TOTAL 3		
Purpose of Program	Computes the chi-square value from a contingency table			Computes the phi coefficient between two variables which are dichotomous			Calculate means, maximums, minimums and standard deviations for a set of data		
Complexity Level	Most	Moderate	Least	Most	Moderate	Least	Most	Moderate	Least
# Assignment Statements	22-26	22-24	23-26	26	26	29-31	26-28	23-24	20-21
# Control Flow Statements	22	16	16	25	24	11	21	20	14
# Other Statements	4	2	4	2	2	2	2	2	2
TOTAL STATEMENTS	48-52	40-42	42-46	53	52	42-44	49-51	45-46	36-37
McCabe's Metric	18	9	9	25	16	6	24	12	8
Halstead's E (thousands of mental discriminations)	98	66	80	160	110	250	140	93	87

Appendix 6.2.2 - A Description of the Engineering Programs

PROGRAM #	BESSEL 4			FOURIER 5			INTEG 6		
Purpose of Program	Computes the Bessel function for a given argument and order			Fourier analysis of a periodically tabulated function			To compute an approximation for an integral by the trapezoidal rule		
Complexity Level	Most	Moderate	Least	Most	Moderate	Least	Most	Moderate	Least
# Assignment Statements	26-28	26-28	26-28	29	24-26	24-26	32	31-33	31-33
# Control Flow Statements	24	27	22	8	10	10	12	15	15
# Other Statements	2	2	2	2	2	2	4	4	4
TOTAL STATEMENTS	52-54	55-57	50-52	39	36-38	36-38	48	50-52	50-52
McCabe's Metric	24	16	14	9	6	5	14	10	10
Halstead's E (thousands of mental discriminations)	110	100	90	43	31	33	75	66	70

Appendix 6.2.3 - A Description of the Non-Numeric Programs

PROGRAM #	SELECT 7			UNIQUE 8			WIDTH 9		
Purpose of Program	Selects a random subset of characters and a test item			Strips adjacent duplicate items			Copies a complete string or selectively copies parts of a string into another string.		
Complexity Level	Most	Moderate	Least	Most	Moderate	Least	Most	Moderate	Least
# Assignment Statements	30	24	25	18	20	14	15	15-16	15-17
# Control Flow Statements	13	15	12	36	27	23	22	17	17
# Other Statements	5	5	5	2	2	2	7	7	8
TOTAL STATEMENTS	48	44	42	56	49	39	44	39-40	40-42
McCabe's Metric	15	10	8	22	15	14	14	9	8
Halstead's E (thousands of mental discriminations)	73	37	31	120	83	39	45	29	26

APPENDIX 6.3.3

SELECT, CONTROL FLOW LEVEL 3, MNEMONIC LEVEL 1

```
      SUBROUTINE SELECT(J,B,M,U,K,P,J1,M1)
      INTEGER B(J), U,M,D(26),Q,P,Y,N(26),L
      EXTERNAL L
      DATA N/1HA,1HB,1HC,1HD,1HE,1HF,1HG,1HH,1HI,1HJ,
     1 1HK,1HL,1HM,1HN,1HO,1HP,1HQ,1HR,1HS,1HT,1HU,1HV,1HW,
     2 1HX,1HY,1HZ/
      IF(J.LE.25) GO TO 90
      P=99
      GO TO 500
90    P=0
      DO 100 I=1,26
      D(I)=N(I)
100   CONTINUE
      DO 120 I=1,25
      Q=L(I,26,J1,M1)
      Y=D(Q)
      D(Q)=D(I)
      D(I)=Y
120   CONTINUE
      DO 140 I=1,J
      B(I)=D(I)
140   CONTINUE
      IF(RAN(J1,M1).GT.0.5) GO TO 200
      M = 1
      K=L(1,J,J1,M1)
      U=B(K)
      GO TO 500
200   K1=J+1
      K=L(K1,26,J1,M1)
      U=D(K)
      M = 2
      K=0
500   RETURN
      END
      INTEGER FUNCTION L(I1, N1, J1,M1)
      INTEGER I1, N1, J1, M1
      IF (N1.GE.I1) GO TO 10
      IY = N1
      N1 = I1
      I1 = IY
 10   R = N1 - I1 + 1
      G = RAN (J1, M1)
      L = IFIX (G * R + FLOAT (I1))
      END
```

APPENDIX 6.3.1

INTEG, CONTROL FLOW LEVEL 1, MNEMONIC LEVEL 3

```
      SUBROUTINE QATR(LBOUND,UBOUND,ABSERR,NDIM,FNCT,RESULT,ERROR,ARRAY)
      DIMENSION ARRAY(NDIM)
      INTEGER ERROR
      REAL LBOUND, INCRE
      ARRAY(1)=.5*(FNCT(LBOUND)+FNCT(UBOUND))
      WIDTH=UBOUND-LBOUND
      IF(NDIM-1)90,90,10
10    IF(WIDTH)20,110,20
20    HAFWID=WIDTH
      ERRMED=ABSERR/ABS(WIDTH)
      CON1=0.
      VAL1=1.
      INDEX3=1
      DO 80 I=2,NDIM
      RESULT=ARRAY(1)
      CON2=CON1
      INCRE=HAFWID
      HAFWID=.5*HAFWID
      VAL1=.5*VAL1
      ARGUM=LBOUND+HAFWID
      DLTX=0.
      DO 30 J=1,INDEX3
      DLTX=DLTX+FNCT(ARGUM)
      ARGUM=ARGUM+INCRE
30    CONTINUE
      ARRAY(I)=.5*ARRAY(I-1)+VAL1*DLTX
      XTRAPL=1.
      INDEX1=I-1
      DO 40 J=1,INDEX1
      INDEX2=I-J
      XTRAPL=4. * XTRAPL
      ARRAY(INDEX2)=ARRAY(INDEX2+1)+(ARRAY(INDEX2+1)-ARRAY(INDEX2))/
     1    (XTRAPL-1.)
40    CONTINUE
      CON1=ABS(RESULT-ARRAY(1))
      IF(I-5)70,50,50
50    IF(CON1-ERRMED)110,110,60
60    IF(CON1-CON2)70,120,120
70    INDEX3=2*INDEX3
80    CONTINUE
90    ERROR=2
100   RESULT=WIDTH*ARRAY(1)
      RETURN
110   ERROR=0
      GO TO 100
120   ERROR=1
      RESULT=WIDTH*RESULT
      RETURN
      END
      FUNCTION FNCT(ARGUM)
      FNCT=1./(2.+ARGUM)
      RETURN
      END
```

APPENDIX 6.3.2

CHISQ, CONTROL FLOW LEVEL 2, MNEMONIC LEVEL 2

```
      SUBROUTINE CHISQ(MAT,N,M,CS,DEG,ERR,RTOT,CTOT)
      INTEGER ERR,DEG,PTR
      REAL MAT
      DIMENSION MAT(100),RTOT(N),CTOT(M)
      NM=N*M
      ERR=0
      CS=0.0
      DEG=(N-1)*(M-1)
      IF (DEG .GT. 0) GO TO 10
      ERR=2
      RETURN
10    DO 20 I=1,N
      RTOT(I)=0.0
      PTR=I-N
      DO 20 J=1,M
      PTR=PTR+N
      RTOT(I)=RTOT(I)+MAT(PTR)
20    CONTINUE
      PTR=0
      DO 30 J=1,M
      CTOT(J)=0.0
      DO 30 I=1,N
      PTR=PTR+1
      CTOT(J)=CTOT(J)+MAT(PTR)
30    CONTINUE
      GTOT=0.0
      DO 40 I=1,N
      GTOT=GTOT+RTOT(I)
40    CONTINUE
      IF (NM .EQ. 4) GO TO 60
      PTR=0
      DO 50 J=1,M
      DO 50 I=1,N
      PTR=PTR+1
      EXPT=RTOT(I)*CTOT(J)/GTOT
      IF (EXPT .LT. 1.0) ERR=1
      CS=CS+(MAT(PTR)-EXPT)*(MAT(PTR)-EXPT)/EXPT
50    CONTINUE
      RETURN
60    CS=GTOT*(ABS(MAT(1)*MAT(4)-MAT(2)*MAT(3))-GTOT/2.0)**2
     1 /(CTOT(1)*CTOT(2)*RTOT(1)*RTOT(2))
70    RETURN
      END
```

APPENDIX 6.4

MEASURING COMPLEXITY OF CONTROL FLOW

In developing a metric for software complexity, one approach might consider the number of statements in a program, thus equating length and complexity. A slightly more sophisticated measure is the percent of statements that affect control flow. A Bell Telephone Laboratories study (Davis, Dickman, Kouni, & Amster, 1976) used this metric on a large number of programs. It has a problem because complexity can be held constant as the size of the program increases.

To assign a metric to control flow complexity, we must examine the elementary control structures of a program. This requires breaking the program down into elementary building blocks, assessing the complexity of each block, and then combining these assessments into higher level components.

Halstead (1975) accomplished this decomposition and synthesis by choosing operands and operators as the smallest conceptual units to develop E, his measure of the complexity of a program.

At a more abstract level, we can define statements and groups of statements which represent cognitive blocks (or chunks) to a programmer (e.g., DO, GO TO). These blocks are probably more representative of the way people manipulate concepts than the smaller units Halstead uses. Ramsey

(1977) is currently involved in experimental studies to test this assertion.

Another attempt to work at this more abstract level is shown by McCabe (1976), who has defined complexity in relation to the decision structure of the program. He ignores the data structure totally. His complexity metric, V(G), is the classical graph-theory cyclomatic number defined as: # edges - # nodes + # connected regions. Simply stated, he counts the number of basic control paths through a computer program.

The simplest program possible would have V(G) = 1. Sequences do not add to the complexity. IF-THEN-ELSE is valued as 2, increasing the complexity by 1, a DO or DO WHILE is also 2, the assumption being that there are really only two control paths, the straight path through the DO and the return to the top, regardless of the number of times executed. Clearly a DO executed 25 times is not 25 times more complex than a DO executed once.

McCabe's method is explained only for structured programs. In order to compute the metric for unstructured programs, several alterations were made. An additional RETURN was counted as an extra path in each case, keeping the cyclomatic number the same as a "GO TO end" would have.

For statements of the form: IF( ) 100, 200, 300, the complexity was increased by 2 as opposed to the logical IF, which increases the complexity by 1. These are small

changes which appear to be reasonable extensions of McCabe's theory. However, one question which arises is the case of the arithmetic IF where two paths are the same:

```
IF ( ) 100, 100, 200
```

Should this add 1 or 2 to the complexity? In order to standardize the procedure, it was counted as the standard arithmetic IF with 2 added to the V(G) metric.

A limitation of McCabe's measure is that it does not deal with an important feature that may affect program complexity. There is no provision for considering the level of nesting in various constructions. For example, the complexity of three DO loops in succession would be rated exactly the same as three DOs that are nested. Possibly at some later time it will be decided that these two conditions have the same complexity, but at this time it seems rash to prematurely exclude nesting as a major contributor of complexity. Presently, many programming shops limit the degree to which nesting is allowed because managers feel it causes problems.

Sullivan and his associates (Bell & Sullivan, 1974; Sullivan, 1973) at the MITRE Corporation, have incorporated the effects of nesting levels into a quantitative measure of complexity. Like McCabe, Sullivan works with a program flow graph, making his metric independent of the programming language used. Sullivan breaks the code into units such that he can define a "local complexity" at any point as the

number of "active concepts" one must consider at that point in the program. He suggests several ways of combining the local complexities into an overall complexity measure for the program. For example, sum the local complexities or take the largest local complexity.

The problem with this metric is that it is complicated to compute and has been implemented only in JOVIAL to scan JOVIAL code. Hand calculation would be extremely tedious and probably error-prone for any non-trivial program. It is not even clear that the decompositions are unique in all cases. Were it discovered to be a good predictor of complexity, it would still take machine implementation in several languages to get people to use it. A similar metric which can be easily computed by machine has recently been described by Richards (1976).

Reiter (1977) has developed a new metric, designed to eliminate the problems discussed above. He uses the same rating scheme for the three basic structures described above, but in addition, assignment statements are accounted for in the complexity metric. He represents a program as a group of nested boxes. Complexity is evaluated from the innermost part of the nest outward, adding a weighting factor for each escape to a higher level. This appears to be a reasonable approach. However, the metric is in the development stage and has not been tested. It is therefore difficult to decide whether the allocation of values to the

assignment statements and to the escape from the nested levels is well chosen.

APPENDIX 6.5.1

MNEMONIC VARIABLE NAMES FOR THE STATISTICAL PROGRAMS

PHI

PROGRAM 1		
M1 LEAST	M2 MEDIUM	M3 MOST
J	N	NUMOBS
F	V1	VECT1
U	V2	VECT2
S	HIVAL1	CODE1
Z	HIVAL2	CODE2
R	ANS	PHI1
H	CS	CHISQ
X	MAX	MAXPHI
M	ERR	NULL
Q	CELL1	A
A	CELL2	B
T	CELL3	C
W	CELL4	D
I	I	I
V	N2	NOBS2
D	PROP1	AMT1
G	PROP2	AMT2
Z	PROP3	AMT3
P	PROP4	ANT4

CHISQ

PROGRAM 2		
M1 LEAST	M2 MEDIUM	M3 MOST
L	NM	TABSIZ
M	ERR	ERRCOD
F	CS	CHISQA
N	DEG	DF
O	RTOT	RTOTAL
K	PTR	IPNTR
T	CTOT	CTOTAL
G	GTOT	TOTAL
H	EXPT	EXPECT
X	MAT	MATRIX
L1	N	NROWS
M1	M	NCOLS
I	I	I
J	J	J
*N1	KTR	KOUNT
*K1	DUM1	DUMMY1
*L2	DUM2	DUMMY2
*Z	DUM3	DUMMY3

* VARIABLES NOT IN C2 VERSIONS

TOTAL

PROGRAM 3		
M1 LEAST	M2 MEDIUM	M3 MOST
X	MAT	MATRIX
T	VEC	VECTOR
Q	AV	AVG
U	SM	SUM
W	SD	STDDEV
V	MN	MINOBS
S	MX	MAXOBS
O	KT	KOUNT
L	VAR	NVAR
N	OBS	NOBS
K	K	K
I	I	I
J	J	J

APPENDIX 6.5.2

MNEMONIC VARIABLE NAMES FOR THE ENGINEERING PROGRAMS

BESSEL

PROGRAM 4		
M1 LEAST	M2 MEDIUM	M3 MOST
P	X	ARG
N	N	ORDER
X	ANS	RESULT
K	ERR	ERROR
B	PI	PI
H	TOL	LIMIT
V	V	VAR
A	T	TERM
D	FJ	FCTJ
S	FI	FCTI
Z	FN	FCTORD
N	I	I
J	J	J

INTEQ

PROGRAM 5		
M1 LEAST	M2 MEDUIM	M3 MOST
Y	LBND	LBOUND
W	UBND	UBOUND
H	TOLR	ABSERR
L	BISCT	NDIM
T	EXFCT	FNCT
S	INTGRL	RESULT
M	ERR	ERROR
V	STOR	ARRAY
O	INTV	WIDTH
Q	HAF	HAFWID
A	ERT	ERRMED
Z	C1	CON1
G	V1	VAL1
N	I3	INDEX3
R	C2	CON2
C	IN1	INCRE
B	X	ARGUM
X	DX	DLTX
D	XTR	XTRAPL
M1	I1	INDEX1
K	I2	INDEX2
I	I	I
J	J	J

FOURIER

PROGRAM 6		
M1 LEAST	M2 MEDIUM	M3 MOST
U	VEC	VECTOR
N	INT1	INTRVL
L	ORD	MAXHAR
F	COS1	COFCOS
T	SIN1	COFSIN
M	ERR	ERROR
Y	INT2	INTVL2
D	S1	SIN1
X	C1	COS1
R	C2	C
H	S2	S
C	F1	FNT1
*K	*IDX	*INDEX
J	J	J
P	RECUR2	U2
Q	RECUR1	U1
I	I	I
Z	RECUR0	U0
S	CS	Q
A	PRD	STEP
V	PIPRD	PISTEP

* NOT IN C1

APPENDIX 6.5.3

MNEMONIC VARIABLE NAMES FOR THE NON-NUMERIC PROGRAMS

SELECT

PROGRAM 7		
M1 LEAST	M2 MEDIUM	M3 MOST
J	N	NCHAR
B	STR	STRING
M	KY	KEY
U	CH	CHAR
K	IND	INDEX
P	ERR	ERROR
D	MIX	SHUFFL
Q	RCHR	NEXT
Y	STO	TEMP
N	ALP	ALPHA
I	I	I
L	RND	RSELCT.
J1	N1	INIT1
M1	N2	INIT2
K1	I1	LOWLIM
I1	L1	LIMIT1
N1	L2	LIMIT2
R	SIZ	SCALE
G	R	RANNO

UNIQUE

PROGRAM 8		
M1 LEAST	M2 MEDIUM	M3 MOST
I	N	NITEMS
N	M	MAXLEN
L	ARR	ITEMS
I1	ALT1	BUFF1
J1	ALT2	BUFF2
M	I2	NSTRIP
K	I1	NORIG
J	L	ITEML
*K1	*LP	*LOOP
**L1	**KG	**KGOTO
*M1	*KTR	*KOUNTR

* NOT IN C1

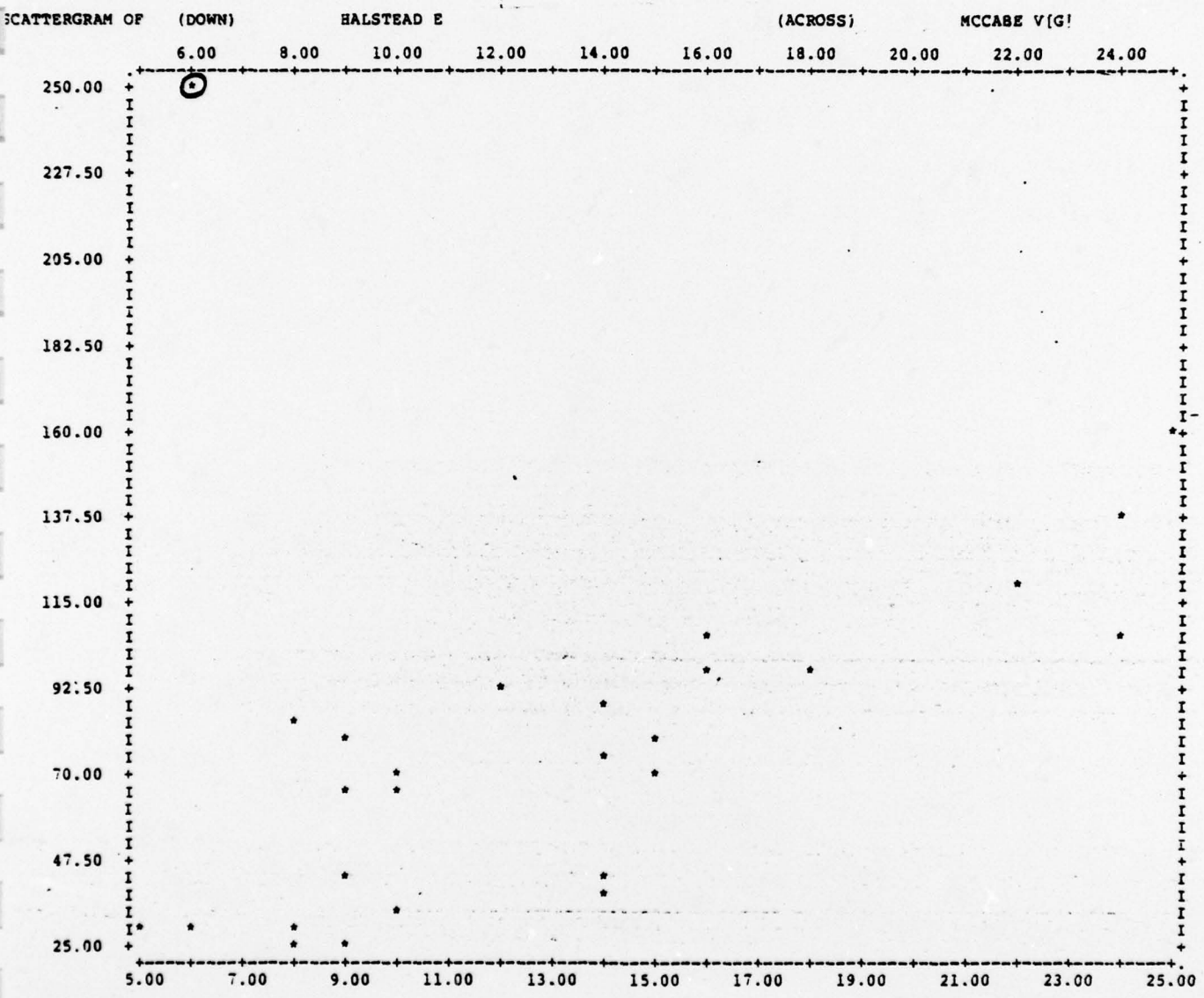
** NOT IN C3 OR C1

WIDTH

PROGRAM 9		
M1 LEAST	M2 MEDIUM	M3 MOST
M1	STR	STRING
J1	LEN	LENGTH
N	WID	WIDTH
M	NSP	NSPACE
J2	LINE	OUTSTR
K2	CTL1	BLANK
L2	CTL2	ZERO
N2	CNTL	CONTRL
K1	NPT	INDEX1
M2	NUM	INDEX2
L1	LI	INDEX3
N1	RCV	STRREC
J3	LOC1	START1
K3	SND	STRSND
M3	LOC2	START2
I1	I1	I1
I2	I2	I2
-	-	*LOOP
I	I	I
J	J	J
K	K	K

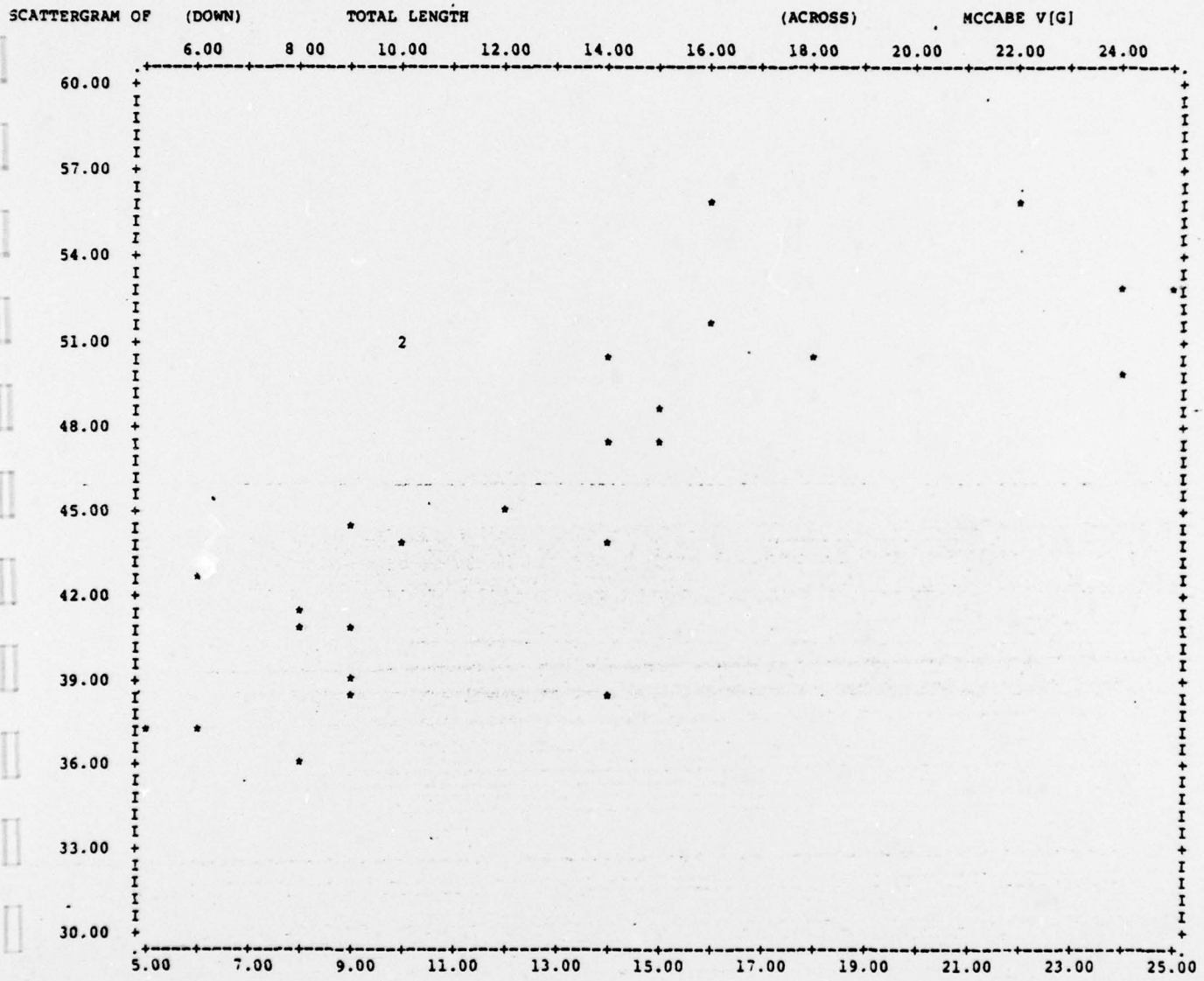
* VERSION C3 ONLY

APPENDIX 6.6.1



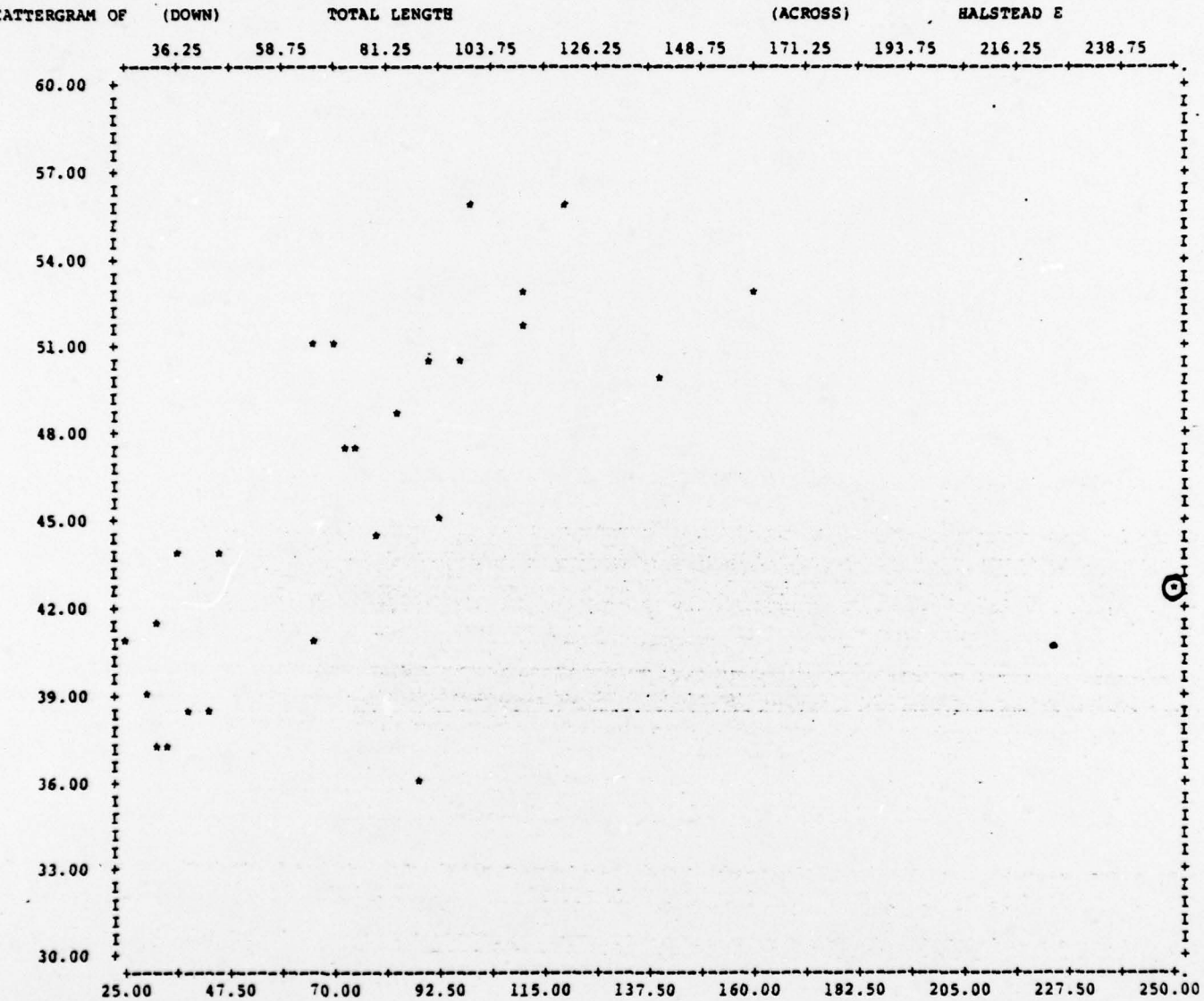
CORRELATION (R) -	.42176	R SQUARED -	.17788	SIGNIFICANCE R -	.01422
STD ERR OF EST -	45.35207	INTERCEPT (A) -	34.05423	STD ERROR OF A -	21.96209
SIGNIFICANCE A -	06678	SLOPE (B) -	3.61582	STD ERROR OF B -	1.55468
SIGNIFICANCE B -	01422				
PLOTTED VALUES -	27	EXCLUDED VALUES-	0	MISSING VALUES -	0

APPENDIX 6.6.2



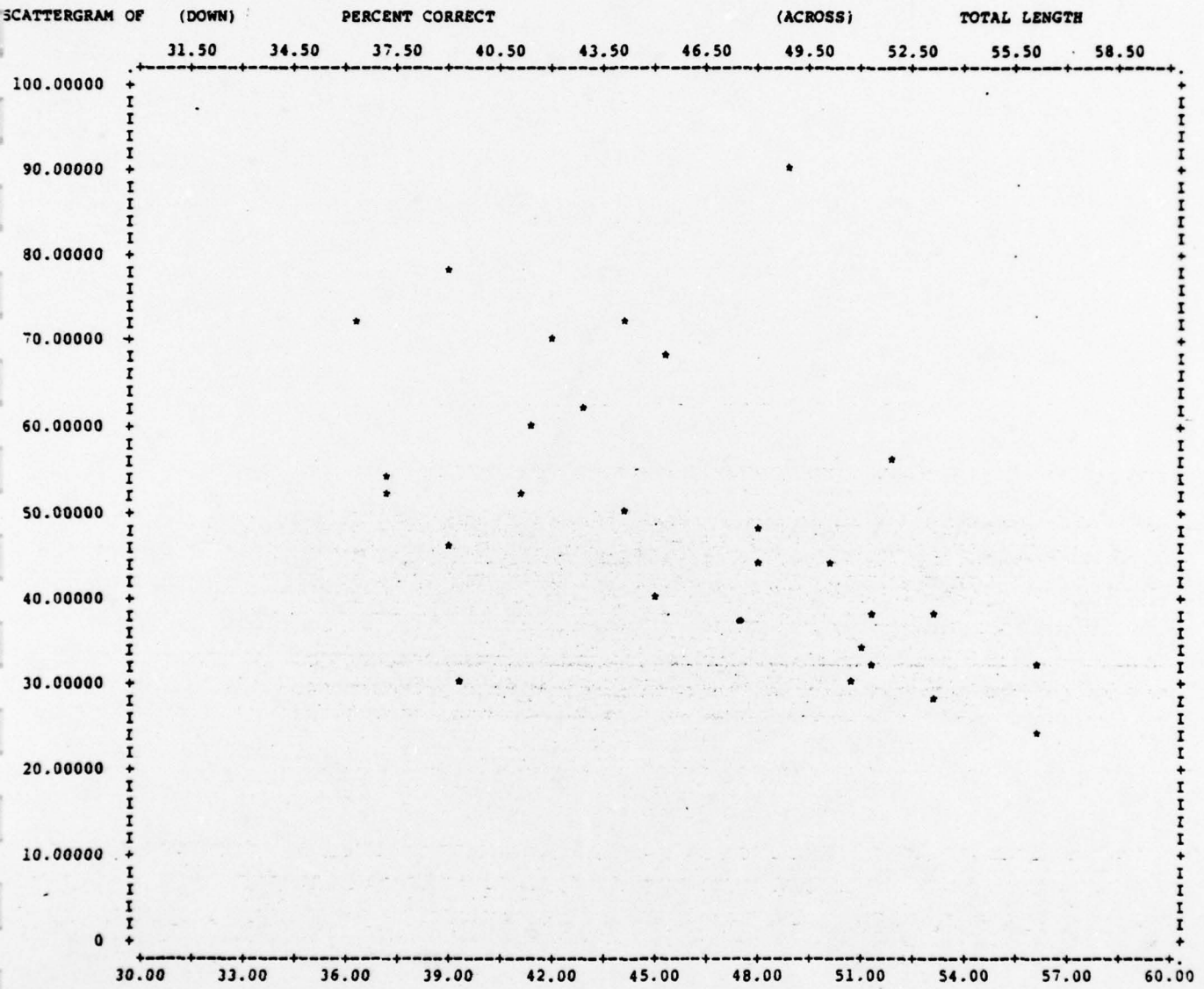
CORRELATION (R) -	.75075	R SQUARED -	.56363	SIGNIFICANCE R -	.00001
STD ERR OF EST -	4.04616	INTERCEPT (A) -	35.83217	STD ERROR OF A -	1.95939
SIGNIFICANCE A -	.00001	SLOPE (B) -	.78818	STD ERROR OF B -	.13870
SIGNIFICANCE B -	.00001				
PLOTTED VALUES -	27	EXCLUDED VALUES -	0	MISSING VALUES -	0

APPENDIX 6.6.3



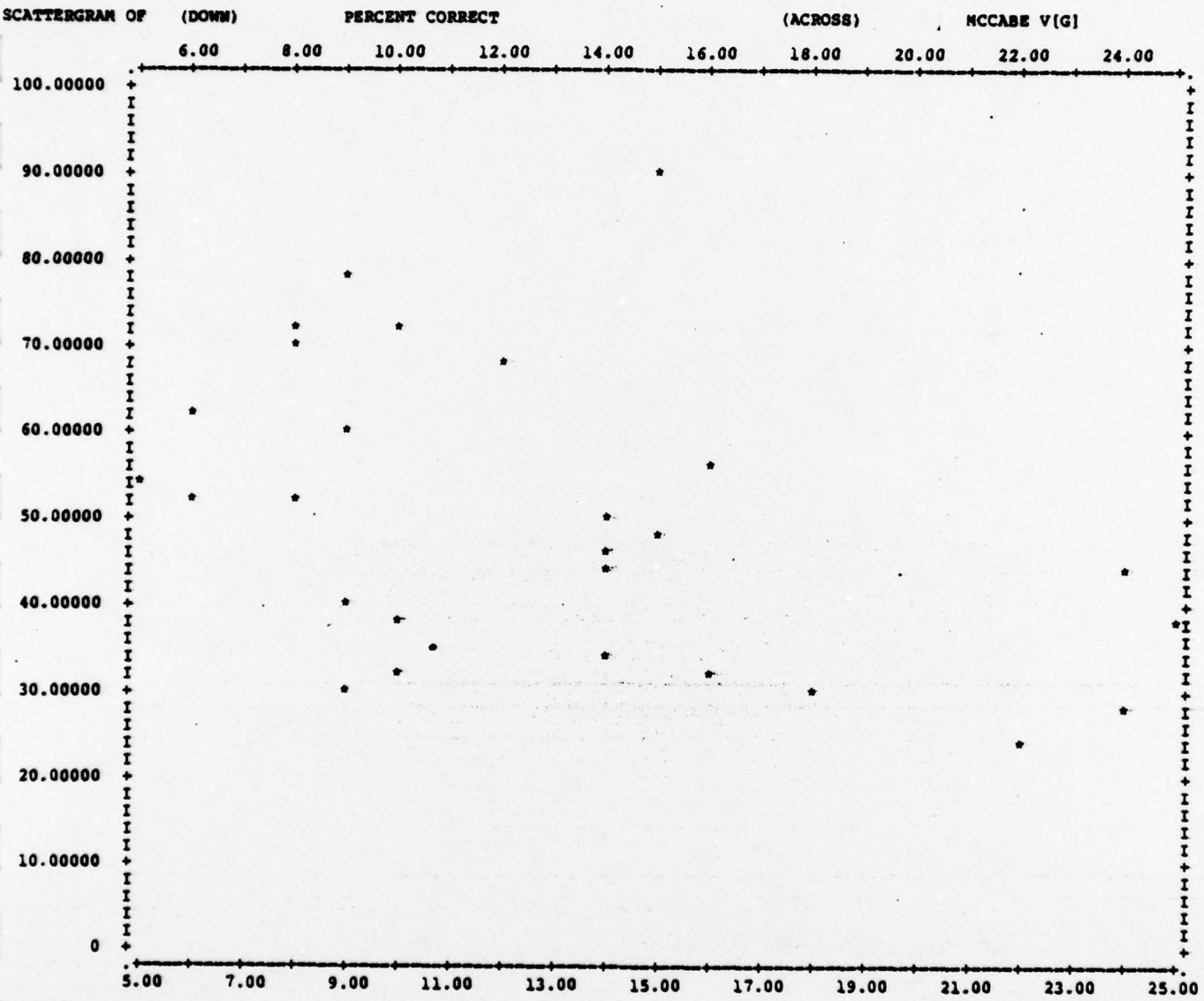
CORRELATION (R) -	.46986	R SQUARED -	.22077	SIGNIFICANCE R -	.00670
STD ERR OF EST -	5.40692	INTERCEPT (A) -	41.39306	STD ERROR OF A -	2.03565
SIGNIFICANCE A -	.00001	SLOPE (B) -	.05754	STD ERROR OF B -	.02162
SIGNIFICANCE B -	.00670				
PLOTTED VALUES -	27	EXCLUDED VALUES-	0	MISSING VALUES -	0

APPENDIX 6.6.4



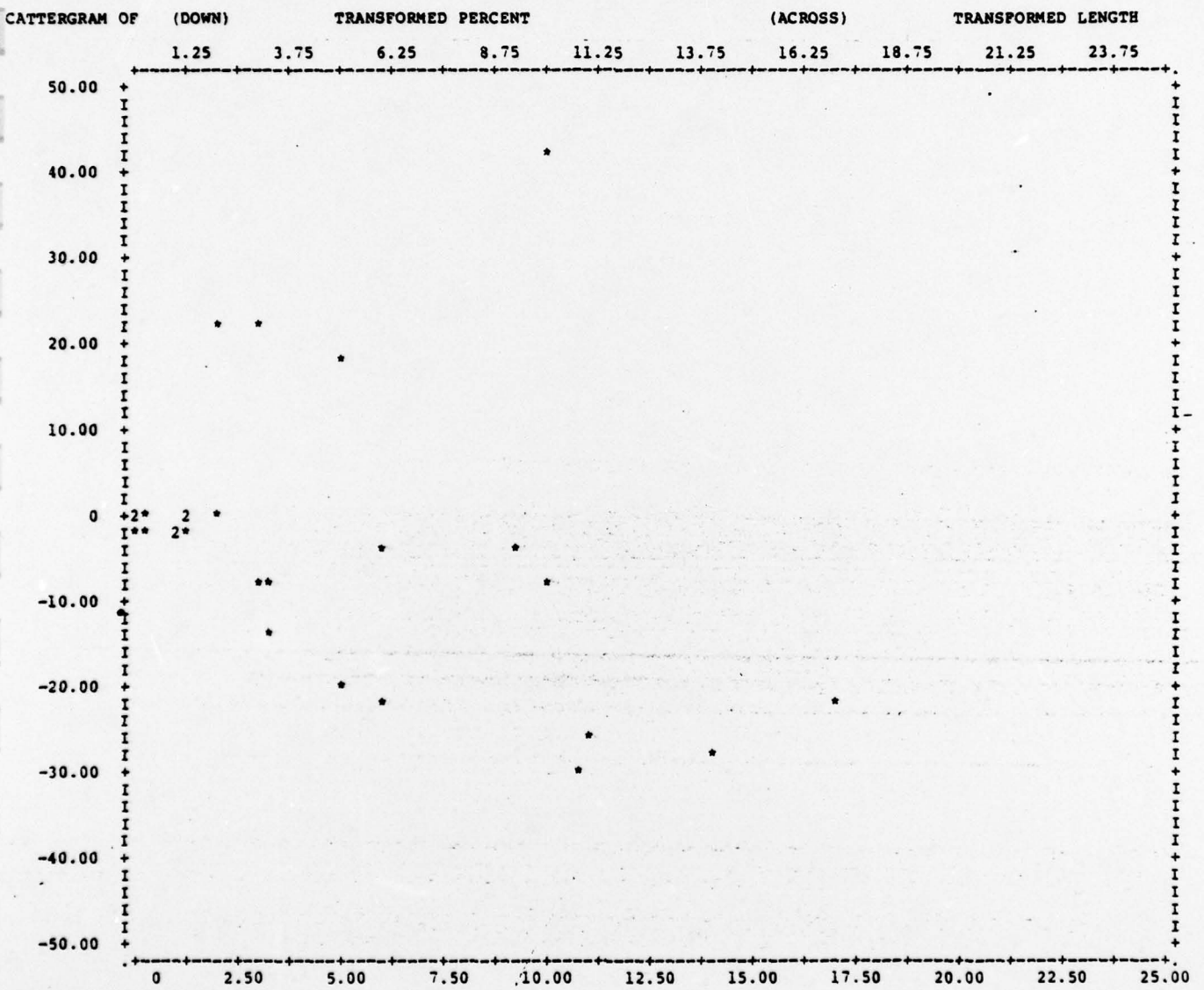
CORRELATION (R) -	-.52643	R SQUARED -	.27713	SIGNIFICANCE R -	.00240
STD ERR OF EST -	14.87754	INTERCEPT (A) -	120.10065	STD ERROR OF A -	22.55263
SIGNIFICANCE A -	.00001	SLOPE (B) -	-1.50394	STD ERROR OF B -	.48579
SIGNIFICANCE B -	.00240				
PLOTTED VALUES -	27	EXCLUDED VALUES -	0	MISSING VALUES -	0

APPENDIX 6.6.5



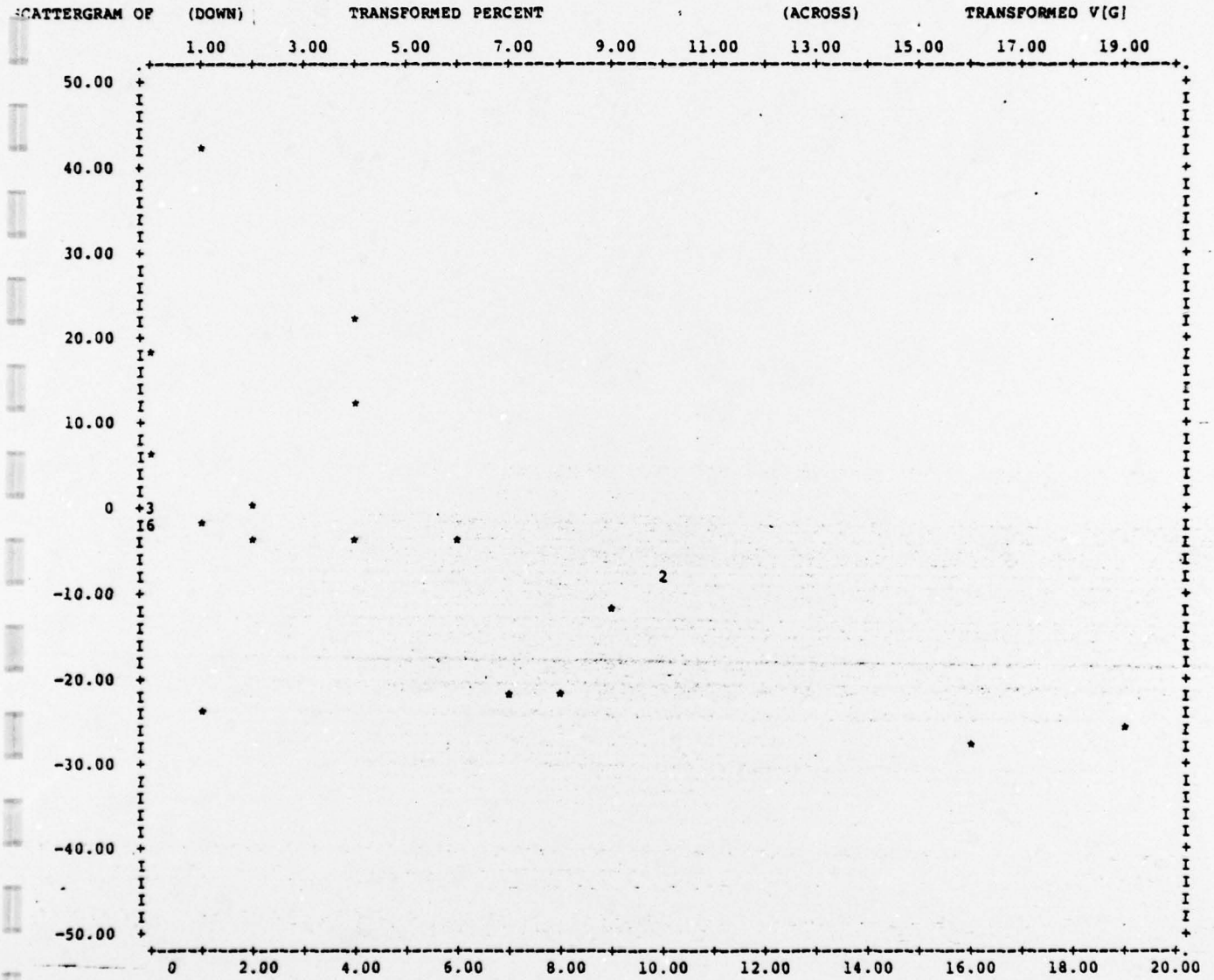
CORRELATION (R)-	-.44948	R SQUARED -	.20203	SIGNIFICANCE R -	.00933
STD ERR OF EST -	15.63129	INTERCEPT (A) -	68.32077	STD ERROR OF A -	7.56957
SIGNIFICANCE A -	.00001	SLOPE (B) -	-1.34811	STD ERROR OF B -	.53585
SIGNIFICANCE B -	.00933				
PLOTTED VALUES -	27	EXCLUDED VALUES-	0	MISSING VALUES -	0

APPENDIX 6.6.6



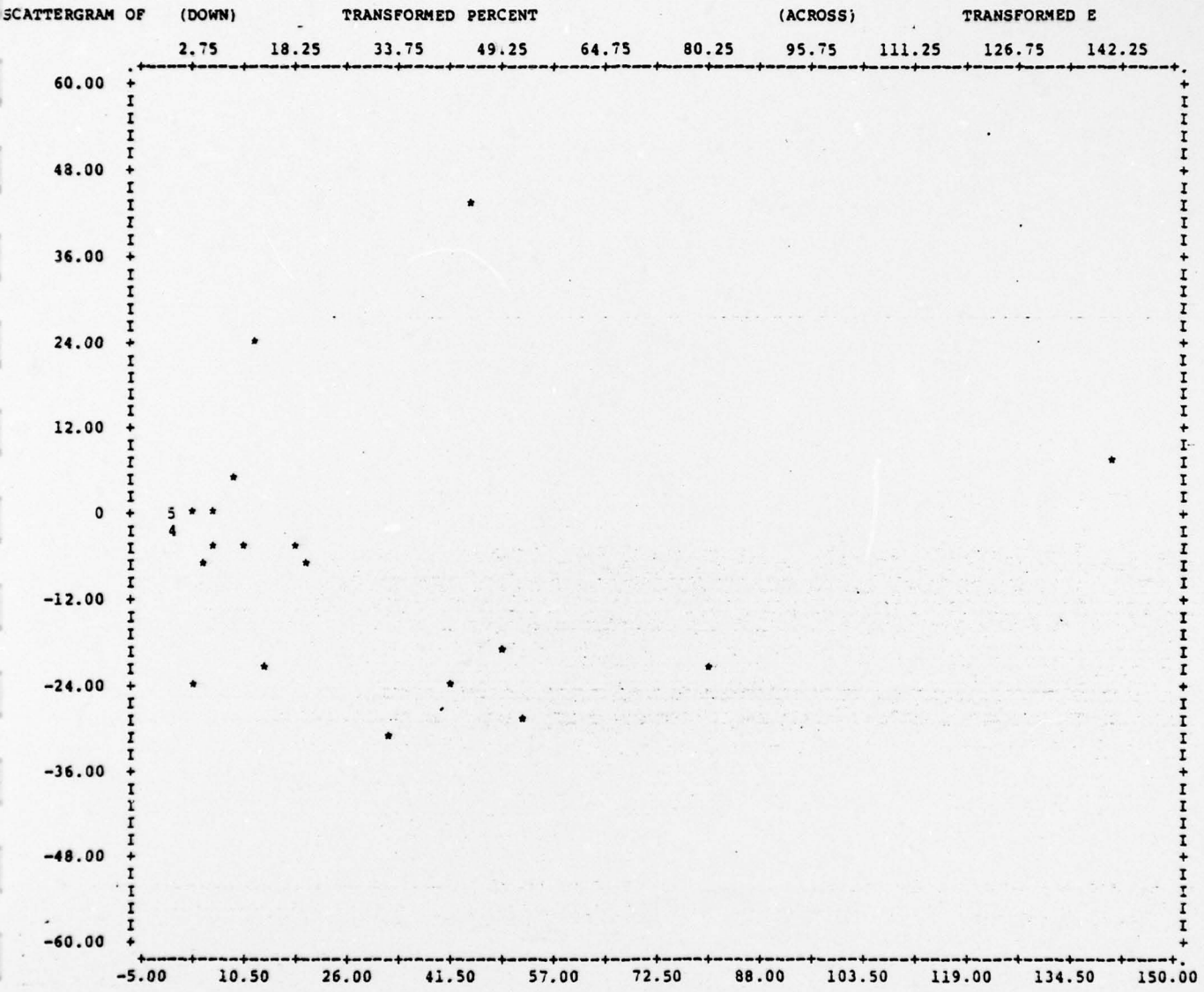
CORRELATION (R) -	-.37695	R SQUARED -	.14210	SIGNIFICANCE R -	.02630
STD ERR OF EST -	15.46792	INTERCEPT (A) -	3.33687	STD ERROR OF A -	4.23920
SIGNIFICANCE A -	.21930	SLOPE (B) -	-1.30229	STD ERROR OF B -	.63998
SIGNIFICANCE B -	.02630				
PLOTTED VALUES -	27	EXCLUDED VALUES-	0	MISSING VALUES -	0

APPENDIX 6.6.7



CORRELATION (R) -	-.56248	R SQUARED -	.31638	SIGNIFICANCE R -	.00113
STD ERR OF EST -	12.79068	INTERCEPT (A) -	4.62167	STD ERROR OF A -	3.08226
SIGNIFICANCE A -	.07314	SLOPE (B) -	-1.63812	STD ERROR OF B -	.48159
SIGNIFICANCE B -	.00113				
PLOTTED VALUES -	27	EXCLUDED VALUES -	0	MISSING VALUES -	0

APPENDIX 6.6.8



CORRELATION (R) -	-.09678	R SQUARED -	.00937	SIGNIFICANCE R -	.31553
STD ERR OF EST -	15.60792	INTERCEPT (A) -	-2.59806	STD ERROR OF A -	3.57718
SIGNIFICANCE A -	.23720	SLOPE (B) -	-.04662	STD ERROR OF B -	.09589
SIGNIFICANCE B -	.31553				
PLOTTED VALUES -	27	EXCLUDED VALUES-	0	MISSING VALUES -	0

OFFICE OF NAVAL RESEARCH, CODE 455
TECHNICAL REPORTS DISTRIBUTION LIST

Director, Engineering Psychology
Programs, Code 455
Office of Naval Research
800 North Quincy Street
Arlington, VA 22217 (5 cys)

Defense Documentation Center
Cameron Station
Alexandria, VA 22314 (12 cys)

Dr. Robert Young
Director, Cybernetics Technology Office
Advanced Research Projects Agency
1400 Wilson Blvd.
Arlington, VA 22209

Office of Naval Research
International Programs
Code 102IP
800 North Quincy Street
Arlington, VA 22217

Director, Information Systems
Program, Code 437
Office of Naval Research
800 North Quincy Street
Arlington, VA 22217

Commanding Officer
ONR Branch Office
ATTN: Dr. J. Lester
495 Summer Street
Boston, MA 02210

Commanding Officer
ONR Branch Office
ATTN: Dr. Charles Davis
536 South Clark Street
Chicago, IL 60605

Dr. Bruce McDonald
Office of Naval Research
Scientific Liaison Group
American Embassy, Room A-407
APO San Francisco, CA 96503

Director, Naval Research Laboratory
Technical Information Division
Code 2627
Washington, D.C. 20375 (6 cys)

Office of the Chief of Naval Operations,
OP 987H
Personnel Logistics Plans
Department of the Navy
Washington, D.C. 20350

Mr. Arnold Rubinstein
Naval Material Command
NAVMAT 0344
Department of the Navy
Washington, D.C. 20360

Commander
Naval Air Systems Command
Human Factors Programs, AIR 340F
Washington, D.C. 20361

Commander
Naval Air Systems Command
Crew Station Design, AIR 5313
Washington, D.C. 20361

Mr. T. Momiyama
Naval Air Systems Command
Advance Concepts Division, AIR 03P34
Washington, D.C. 20361

Commander
Naval Electronics Systems Command
Human Factors Engineering Branch
Code 4701
Washington, D.C. 20360

Dr. James Curtin
Naval Sea Systems Command
Personnel & Training Analyses Office
NAVSEA 074C1
Washington, D.C. 20362

Director
Behavioral Sciences Department
Naval Medical Research Institute
Bethesda, MD 20014

Dr. George Moeller
Human Factors Engineering Branch
Submarine Medical Research Laboratory
Naval Submarine Base
Groton, CT 06340

Mr. Phillip Andrews
Naval Sea Systems Command
NAVSEA 0341
Washington, D.C. 20362

Bureau of Naval Personnel
Special Assistant for Research
Liaison
PERS-OR
Washington, D.C. 20370

Navay Personnel Research and
Development Center
Management Support Department
Code 210
San Diego, CA 92152

Dr. Fred Muckler
Navy Personnel Research and
Development Center
Manned Systems Design, Code 311
San Diego, CA 92152

Mr. A.V. Anderson
Navy Personnel Research and
Development Center
Code 302
San Diego, CA 92152

LCDR P.M. Curran
Human Factors Engineering Branch
Crew Systems Department, Code 4021
Naval Air Development Center
Johnsville
Warminster, PA 18950

LCDR William Moroney
Human Factors Engineering Branch
Code 1226
Pacific Missile Test Center
Point Mugu, CA 93042

Human Factors Section
Systems Engineering Test Directorate
U.S. Naval Air Test Center
Patuxent River, MD 20670

Dr. John Silva
Man-System Interaction Division
Code 823, Naval Ocean Systems Center
San Diego, CA 92152

Human Factor Engineering Branch
Naval Ship Research and Development
Center, Annapolis Division
Annapolis, MD 21402

Naval Training Equipment Center
ATTN: Technical Library
Orlando, FL 32813

Dr. Alfred F. Smode
Training Analysis and Evaluation Group
Naval Training Equipment Center
Code N-00T
Orlando, FL 32813

Dr. Gary Poock
Operations Research Department
Naval Postgraduate School
Monterey, CA 93940

Dr. A. L. Slafkosky
Scientific Advisor
Commandant of the Marine Corps
Code RD-1
Washington, D.C. 20380

Mr. J. Barber
Headquarters, Department of the Army,
DAPE-PBR
Washington, D.C. 20546

Dr. Joseph Zeidner
Director, Organization and Systems
Research Laboratory
U.S. Army Research Institute
5001 Eisenhower Avenue
Alexandria, VA 22333

Dr. Edgar M. Johnson
Organization and Systems Research
Laboratory
U.S. Army Research Lab
5001 Eisenhower Avenue
Alexandria, VA 22333

Technical Director
U.S. Army Human Engineering Labs
Aberdeen Proving Ground
Aberdeen, MD 21005

U.S. Air Force Office of Scientific Research
Life Sciences Directorate, NL
Bolling Air Force Base
Washington, D.C. 20332

Dr. Donald A. Topmiller
Chief, Systems Engineering Branch
Human Engineering Division
USAF AMRL/HES
Wright-Patterson AFB, OH 45433

Lt. Col. Joseph A. Birt
Human Engineering Division
Aerospace Medical Research Laboratory
Wright Patterson AFB, OH 45433

Air University Library
Maxwell Air Force Base, AL 36112

Dr. Arthur I. Siegel
Applied Psychological Services, Inc.
404 East Lancaster Street
Wayne, PA 19087

Dr. Gershon Weltman
Perceptronics, Inc.
6271 Variel Avenue
Woodland Hills, CA 91364

Dr. Edward R. Jones
McDonnell-Douglas Astronautics Company - East
St. Louis, MO 63166

Dr. H. Rudy Ramsey
Science Applications, Inc.
40 Denver Technological Center West
7935 East Prentice Avenue
Englewood, CO 80110

Dr. Meredith Crawford
5606 Montgomery Street
Chevy Chase, MD 20015

Dr. Jesse Orlansky
Institute for Defense Analyses
400 Army-Navy Drive
Arlington, VA 22202

Dr. Stanley Deutsch
Office of Life Sciences
HQS, NASA
600 Independence Avenue
Washington, D.C. 20546

Director, National Security Agency
ATTN: Dr. Douglas Cope
Code R51
Ft. George G. Meade, MD 20755

Journal Supplement Abstract Service
American Psychological Association
1200 17th Street, NW
Washington, D.C. 20036 (3 cys)

Dr. William A. McClelland
Human Resources Research Office
300 N. Washington Street
Alexandria, VA 22314

Kin B. Thompson
NDAC
Pentagon, Room BD770
Washington,D.C. 20301

A. Stoholm
NPRDC
San Diego, CA 92152

Director, Human Factors Wing
Defense & Civil Institute of Environmental Medicine
Post Office Box 2000
Downsville, Toronto, Ontario
CANADA

Dr. A.D. Baddeley
Director, Applied Psychology Unit
Medical Research Council
15 Chaucer Road
Cambridge, CB2 2EF
ENGLAND